# Representation of Diffusion Models with applications to downstream tasks

rehan0900

December 2024

## Contents

# 1 Representation learning

## 1.1 To-Do

1. Basically, you want to focus on learning disentangled representations.
2. Shouldn't we think of representation as capturing the manifold of the data?
3. What would it mean for the T2I Diffusion model to be "off-the-manifold"? If I state some gibberish, it would obviously not perform well.
4. How is image-editing related to manifold learning? This is a subsection that you should make. Naturally, you should point out that diffusion models capture the representation h by defining a lower-dimensioanl manifold. Now, if we are able to find meaningful variations along h (those tangential to it), we would have found a space for natural edits and vice versa (orthogonal variations would not point to areas of natural edits). Not only that, if we are able to capture the underlying manifold, we might be able to reconstruct corrupted regions more meaningfully.
5. Thinking in context of embeddings, can we capture the manifold that outputs the most information?
6. Incorporate Manifold Preserving Guided Diffusion 7. Incorporate Riemannian Diffusion Models

## 1.2 Overview

**Representation learning** [6] refers to the process of discovering representations of data that simplify the extraction of meaningful information. In probabilistic approaches, effective representations often reflect the posterior distribution of the key explanatory factors underlying the observed data. Additionally, a useful representation serves as valuable input for supervised prediction tasks. Among the various techniques for learning representations, **deep learning methods** involve the sequential application of multiple non-linear transformations aimed at producing increasingly abstract — and ultimately more effective — data representations.

More formally, if the input data $x$ lives in a high-dimensional space $\mathcal{X}$, then representation learning seeks to map $x$ into a new space $\mathcal{Z}$ where the representation $z$ of the data is more useful for downstream tasks such as classification and regression. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Z}$, where:

$$z = f(x; \theta), \tag{1}$$

where $x$ is the raw input data (e.g., pixels, words, sensor values), $z$ is the learned representation and $\theta$ are the parameters of the model.

Traditionally, deep learning involves learning representations are learned through neural networks. Each layer of a neural network transforms the input $x$ into a new representation $h$ using nonlinear transformations. For a layer $l$, the transformation is:

$$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)}), \tag{2}$$

where, $h^{(l)}$ is the representation at layer $l$ and $\sigma$ is a nonlinear activation function. The transformations are compositional, and thus the learning is hierarchical:

$$z = f_L(f_{L-1}(\dots f_1(x; \theta_1) \dots ; \theta_{L-1}); \theta_L), \tag{3}$$

where each $f_l$ represents a layer transformation with its own parameters $\theta_l$.

The representations $z$ are learned by optimizing a loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}}\left[\ell(f(x;\theta), y)\right], \tag{4}$$

where $\ell$ is the task-specific loss function (e.g., cross-entropy for classification), $f(x;\theta)$ produces predictions or embeddings based on learned representations and $y$ is the ground truth label. During training, the network adjusts $\theta$ such that the learned representations $z$ are optimal for the task.

For **generative models**, representation learning focuses on discovering latent factors $z$ that generate the observed data $x$:

$$p(x, z) = p(x \mid z)p(z), \tag{5}$$

where $p(x \mid z)$ represents the conditional likelihood of the data given latent variables and $p(z)$ is the prior distribution over the latent variables.

A good representation $z$ efficiently encodes the structure of $x$ while disentangling independent factors of variation. For inference, we estimate the posterior $p(z \mid x)$, which often requires approximations in complex settings. The posterior distribution $p(z \mid x)$ determines the latent factors $z$ given observed data $x$. Using Bayes' theorem, we get:

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}, \tag{6}$$

where $p(x)$ is the marginal likelihood or evidence, defined as:

$$p(x) = \int p(x \mid z)p(z)\, dz. \tag{7}$$

This integral sums over all possible latent variables $z$ to compute the likelihood of $x$. The posterior $p(z \mid x)$ combines the prior $p(z)$ and likelihood $p(x \mid z)$, balancing prior assumptions with evidence from the data. However, for complex models, the integral $\int p(x \mid z)p(z)\, dz$ is intractable, making direct computation of $p(z \mid x)$ infeasible.

The goal of representation learning is to optimize the model parameters such that:

1. The latent variables $z$ capture the underlying structure of the data $x$.

2. The posterior $p(z \mid x)$ disentangles independent factors of variation.

The traditional way in order to learn the model parameters (e.g., in $p(x \mid z)$ and $p(z)$) is to maximize the marginal likelihood of the data $x$:

$$\log p(x) = \log \int p(x \mid z)p(z)\, dz. \tag{8}$$

Direct maximization is often difficult because of the integral over $z$. The most common solutions are **variational inference** in which you approximate $p(z \mid x)$ with a simpler distribution $q(z \mid x)$ and **Monte Carlo Methods** which uses sampling techniques to approximate the integral.

## 1.3 Manifolds and Representation

An elegant way to understand a representation of a model is through manifold hypothesis. The manifold hypothesis states that real-world high-dimensional data lies on a low-dimensional manifold embedded in the high-dimensional space. Mathematically, let the data be $x \in \mathbb{R}^D$ where D is large. Then, the data can be well-approximated by a much lower-dimensional space a manifold $\mathcal{M} \subseteq \mathbb{R}^D$ where $\dim(\mathcal{M}) = d \ll D$.

Thus, for any "realistic" input x , there exists a latent representation h such that:

$$x \approx g_\phi(h) \text{ where h} \in \mathbb{R}^D, \dim(\mathcal{M}) = \text{d} \tag{9}$$

When we say that an input $x$ ' is "off the manifold", it means $x$ ' does not lie close to the learned manifold $\mathcal{M}$. Consequently, there is no corresponding latent representation h $\in \mathbb{R}^d$ such that the condition $x' \approx g_\phi(h)$ is satisfied.

Most machine learning algorithms assume inputs are sampled from the data manifold $\mathcal{M}$ where they were trained. If the input lies far from $\mathcal{M}$, unusual or unexpected behavior can occur. Formally, a Manifold is a topological space that locally resembles Euclidean space near each point. Globally, the manifold can have a complex shape or curvature. For example, although a 2D sphere $S^2 \subset \mathbb{R}^3$ is not globally flat, but each region on the sphere resembles $\mathbb{R}^2$.

The criterion that a manifold must locally resemble a Euclidean Space is to ensure that the rules of calculus carry over. In particular, it ensures:

1. **Existence of Tangent Spaces** At each point p, the manifold locally behaves like $\mathbb{R}^d$, so we can define a tangent space to it (much like tangent spaces define directional derivatives in 2D)

2. **Criterion of Smooth Maps** Functions defined on manifolds can be analyzed locally using smooth coordinate charts.

Manifolds are characterized by **tangent spaces**. They specify how x can change while staying on manifold. At a point **x** on a d-dimensional manifold, the tangent plane is given by basis vectors that span the local directions of variation allowed on the manifold. More formally, if $\mathcal{N}$ is defined implicitly as a smooth mapping x = $g(h)$, the Jacobian of g, $J_g$ describes how the manifold is locally oritented:

$$J_g(h) = \frac{\partial g(h)}{\partial h} \in \mathbb{R}^{D \times d} \tag{10}$$

For example, for a sphere, the smooth mapping $g : \mathbb{R}^2 \to \mathbb{R}^3$ is:

$$g(h) = g(\theta, \phi) = \begin{bmatrix} r\sin(\phi)\cos(\theta) \\ r\sin(\phi)\sin(\theta) \\ r\cos(\phi) \end{bmatrix}$$

The Jacobian matrix of this is:

$$J_g(\theta, \phi) = \begin{bmatrix} \frac{\partial g_1}{\partial \theta} & \frac{\partial g_1}{\partial \phi} \\ \frac{\partial g_2}{\partial \theta} & \frac{\partial g_2}{\partial \phi} \\ \frac{\partial g_3}{\partial \theta} & \frac{\partial g_3}{\partial \phi} \end{bmatrix} = \begin{bmatrix} -r\sin(\phi)\sin(\theta) & r\cos(\phi)\cos(\theta) \\ r\sin(\phi)\cos(\theta) & r\cos(\phi)\sin(\theta) \\ 0 & -r\sin(\phi) \end{bmatrix}$$

The columns of the Jacobian matrix are the basis vectors of the tangent space at the point x :

- $\frac{\partial g}{\partial \theta}$ describes the direction of change when $\theta$ varies (moving along the azimuthal angle).

- $\frac{\partial g}{\partial \phi}$ describes the direction of change when $\phi$ varies (moving along the azimuthal angle).

To understand the above in context of example, take the MNIST data. Each data point $\mathbf{x}$ lies on a low-dimensional manifold. The tangent space at $\mathbf{x}$ provides the directions of allowable variation (e.g., rotations, translations, or deformations of the digit). The Jacobian of a learned mapping $g : \mathbb{R}^d \rightarrow \mathbb{R}^D$ describes these directions. The key insight is that manifolds locally behave like flat planes and the Jacobian provides the link between the low-dimensional intrinsic structure and the high-dimensional ambient space.

Let us now get a sense of **orthogonal spaces** to manifolds. Say that the lower dimensional representation that we are learning is defined by $g : h \rightarrow x$ where $g : \mathbb{R}^d \rightarrow \mathbb{R}^D$ and $J_g(h) = \frac{\partial g(h)}{\partial h} \in \mathbb{R}^{Dxd}$ spans the tangential space to the manifold. As explained, movements along the tangent space correspond to meaningful variations on the manifold and thus affect h which is the lower-dimensional representation of data. The normal space $N_x\mathcal{M}$ is the ( $D-d$ ) dimensional complement of the tangent space in $\mathbb{R}^D$. Movements orthogonal to the manifold correspond to deviations that are not part of the manifold and therefore do not affect h. For the case of our sphere, tangential directions (along the surface) correspond to changes in the angles $\theta$ and $\phi$, which are meaningful. The representation g that we would learn would only capture the meaningful changes in $\theta$ and $\phi$ and ignores variations orthogonal to the sphere's surface.

Thus, given an arbitrary point $h \in \mathbb{R}^d$, the data-point x $\in \mathbb{R}^D$ can be decomposed into a tangential component $x_T$ and an orthogonal component $x_\perp$ with respect to the data manifold. The movements along the orthogonal component $x_\perp$ do not capture the manifold that our representation $g$ has learned. Thus, they do not effect h. On the other hand, movements along the tangential component $x_T$ capture the manifold.

More formally, given the Jacobian $J_g(h) = \frac{\partial g(h)}{\partial h}$, our representation must ignore the variations orthogonal to the manifold while not that which is along the manifold:

$$\begin{aligned} \frac{\partial g(h)}{\partial h} \cdot x_\perp = 0 \\ \frac{\partial g(h)}{\partial h} \cdot x_T \neq 0 \end{aligned} \tag{11}$$

To understand this in context of an example, suppose we have a representation (say, an autoencoder), which learns to capture and encode the variations in the latent representation h for a specific digit image x (e.g, a " 3 "). Now, variations along the manifold are changes that correspond to meaningful variations of the digit. For example, making the stroke thicker or thinner, rotating the digit slightly or changing its style (e.g., curvy vs. angular). These variations stay on the manifold because they still represent a valid "3". On the other hand, variations along the manifold do not correspond to meaningful digits. These include adding random noise to the pixels or perturbing parts of the image in a way that doesn't look like a "3" anymore (e.g., stray dots or blurs). These variations move the image off the manifold.

To understand how examining the representation of models using **manifold learning** can provide us practical insights into their workings, lets take the example of **autoencoders**. We can think of autoencoder as projecting the data into a lower-dimensional representation (latent space) and reconstructing it back to effectively capture the structure of the data manifold.

Let the data be $x \in \mathbb{R}^D$ where D is large. In autoencoders, an encoder is responsible for mapping the input data x to a lower-dimensional representation h : $f_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^d$ while a decoder is responsible for reconstructing the input x from its lower dimensional representation h : $g_\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$. The reconstructed input is:

$$x' = g_\phi\left(f_\theta(x)\right) \tag{12}$$

The objective of the autoencoder is to minimize the reconstruction error:

$$L_{rec} = \|x - g_\phi\left(f_\theta(x)\right)\|^2 \tag{13}$$

The lower-dimensional representation h learned by the encoder is often referred to as the "latent space" or "encoding space." This space captures the essential features of the input data. Now, given that we have been able to capture a lower-dimensional manifold d using autoencoders, there are two types of possible variations with respect to the original data x that the autoencoder can handle.For the tangential Variations (On the Manifold), the autoencoder learns to capture and encode these variations in the latent representation h. For example, if you input an image of "3" with slightly thicker strokes, the latent representation $h$ will reflect this meaningful variation. The reconstruction $f(x)$ will match the input closely because it stays on the manifold. If noise or irrelevant perturbations are added to the image (e.g., a speck of noise in the background), the autoencoder projects the input back onto the manifold. This means the autoencoder "corrects" the image, ignoring the noise, and outputs a clean version of the "3."

The thing is, equation 13 focuses solely on minimizing the reconstruction error without any attempt at capturing the underlying patterns of the data. The autoencoder learns to reconstruct not just the manifold structure, but also the irrelevant directions (orthogonal to the manifold). This means that the reconstruction error is minimized even for noisy or off-manifold points. The latent representation h = $f_\theta(x)$ becomes sensitive to variations orthogonal to the manifold. As a result without regularization, the latent representation h is not constrainted to be smooth or aligned with the intrinsic manifold directions.

We can formalize this in context of the Jacabian of the manifold. Let $x_{clean}$ be the clean input and $x_{\text{noisy}} = x_{\text{clean}} + \delta x$ the noisy input. If the encoder $f_\theta$ is sensitive to small changes in x , then a small amount of noise causes the latent representation h to change drastically. The noisy input is mapped far from the clean input in the latent space, even though they look visually similar. Now, if the encoder is sensitive to small changes in x , then the norm $\|J_g(h)\|$ of the jacobian is large.

To mitigate the problem of autoencoders overfitting and not capturing the data manifold, Contractive Autoencoders (CAE) penalize the Jacobian norm in the loss function:

$$L_{CAE} = \|x - g_\phi\left(f_\theta(x)\right)\|^2 + \lambda \|J_f(x)\|_F^2 \tag{14}$$

The Jacobian penalty ensures that small input perturbations cause minimal changes in the latent representation. More formally, a small neighborhood of points around an input $x$ in the input space $\mathbb{R}^D$ is mapped to a smaller neighborhood in the latent space or output space $\mathbb{R}^d$. This causes the input space to "contract" or "shrink" locally into a smaller, smoother region in the latent space.

The sensitivity is measured by the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input:

$$\|J_f(\vec{x})\| = \sum_{ij}\left(\frac{\partial h_j(\vec{x})}{\partial x_i}\right)^2 \tag{15}$$

Where $h_j$ is one unit output in the compressed code $\vec{z} = f(x)$. This penalty term is the sum of squares of all partial derivatives of the learned encoding with respect to input dimensions. The authors claimed that empirically this penalty was found to carve a representation that corresponds to a lower-dimensional non-linear manifold, while staying more invariant to majority directions orthogonal to the manifold.

Another solution to avoid overfitting was introduced in [32]. They proposed a modification to the basic autoencoder in which the input is partially corrupted by adding noises to or masking some values of the input vector in a stochastic manner,

$$\tilde{x} \sim q_D(\tilde{x} \mid \vec{x}) \tag{16}$$

Then the model is trained to recover the original input:

$$\begin{gathered}\tilde{x}^{(i)} \sim q_D\left(\tilde{x}^{(i)} \mid \vec{x}^{(i)}\right) \\ L_{\mathrm{DAE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(\vec{x}^{(i)} - g_\phi\left(f_\theta\left(x^{(i)}\right)\right)\right)^2\end{gathered} \tag{17}$$

Where $\mathcal{M}_D$ defines the mapping from the true data samples to the noisy or corrupted ones. For each input, a fixed number $v$ d of components are chosen at random, and their value is forced to 0 , while the others are left untouched. All information about the chosen components is thus removed from that particuler input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks".

The authors explain why denoising autoencoders learn a more effective representation than standard autoencoders by invoking the **manifold learning** perspective: a corrupted sample lies farther from the data manifold compared to uncorrupted samples. Consequently, the stochastic mapping from the corrupted input $\tilde{x}$ to the reconstructed data $x'$ must learn to make larger adjustments to bring the corrupted input closer to the manifold. In the limit, the operator should map even distant points to a small region near the manifold.

# 2 Image Editing

## 2.1 To Do

- In your overview, you should talk about the problems that works in Image Editing aim to resolve. For example, you should talk in detail about the problem of **catastrophic forgetting** and **attribute-binding** and so on. Furthermore, you need to incorporate how exactly does image-editing relate with the representation of Diffusion models on a deeper level. - When it comes to image editing, you really lost the tree for leaves. Your aim should be to target niche works. I just feel that with so much work out there on the subject, there needs to be a realization of some very particular topic . Can't just add to the weight of all this bs that has been published.

## 2.2 Overview

Image editing can be described as the process of transforming an input image $\mathbf{I}_{\text{in}} \in \mathbb{R}^{H \times W \times C}$ into an output image $\mathbf{I}_{\text{out}} \in \mathbb{R}^{H' \times W' \times C'}$. This transformation is governed by a function $f$ which applies the desired modifications to the input image based on a set of parameters $\mathbf{P}$. Formally, the process is represented as:

$$\mathbf{I}_{\text{out}} = f(\mathbf{I}_{\text{in}}, \mathbf{P}) \tag{18}$$

The function $f$ encapsulates a variety of editing operations, ranging from simple pixel-level adjustments to complex semantic modifications. Pixel-level editing involves direct manipulation of the image's pixel values. For example, adjusting the brightness or contrast of an image can be expressed as:

$$\mathbf{I}_{\text{out}}(i, j, c) = g(\mathbf{I}_{\text{in}}(i, j, c), \mathbf{P}) \tag{19}$$

where $g$ modifies the intensity of pixel $(i, j)$ in channel $c$. A practical example is applying a grayscale filter to an image, where each pixel's intensity is recalculated as a weighted sum of the original RGB values. Similarly, geometric transformations, such as scaling, rotation, and translation, alter the spatial arrangement of pixels. These transformations can be modeled using a transformation matrix $M$, where:

$$\mathbf{I}_{\text{out}}(x', y') = \mathbf{I}_{\text{in}}(M^{-1} \cdot [x', y', 1]^T) \tag{20}$$

For instance, rotating an image by 90 degrees involves modifying the coordinates of each pixel according to the rotation matrix.

More advanced forms of image editing involve semantic modifications, where high-level content, such as object shapes or relationships, is altered. This often requires working in a latent space. For example, an input image is encoded into a latent representation $\mathbf{z}_{\text{in}}$, which is then modified using a function $h(\mathbf{z}_{\text{in}}, \mathbf{P})$ to produce $\mathbf{z}_{\text{edit}}$. The edited latent representation is decoded back into the output image as:

$$\mathbf{I}_{\text{out}} = D(\mathbf{z}_{\text{edit}}). \tag{21}$$

A practical example of semantic editing is changing the color of an object in the image or modifying facial expressions in a portrait by manipulating the latent vectors.

Image editing is intimately connected to **Representation learning**. Representation learning often aims to disentangle different factors of variation in an image, such as pose, color, texture, or object identity. This disentanglement makes it possible to edit specific attributes independently without affecting others. For example, in an image of a car, disentangled representations can allow editing the color of the car while preserving its shape and background. Another example is the task of generating images of human faces. Each image $x$ can be explained by a set of latent factors $z$ that correspond to distinct attributes of the face. For example:

- $z_1$: Lighting condition (e.g., bright or dark).
- $z_2$: Pose of the head (e.g., straight, left tilt, right tilt).
- $z_3$: Hair color (e.g., black, blonde, brown).
- $z_4$: Smile intensity (e.g., neutral, slight smile, wide smile).

These factors are independent of each other in the real world. Changing the hair color should not influence the smile, and adjusting the lighting should not alter the head pose. A good representation $z$ should disentangle these independent factors such that each dimension $z_i$ in the latent space corresponds to one meaningful variation in the data $x$. If we are able to obtain these underlying representations, we can easily disentangle objects of interest.

## 2.3 Spa-Text

In traditional models like Stable Diffusion, the user can specify a global scene using text (e.g., ”a sunny day at the beach”), but the exact placement and layout of objects are unpredictable. Spatial relationships between objects (e.g., ”the ball near the Labrador’s paw”) are often ignored or inaccurately interpreted. Furthermore, methodologies that allow spatial control often rely on predefined labels such as ”dog” or ”tree”. These approaches cannot describe nuanced object characteristics like ”a Labrador wearing a red collar” or ”a small blue ball.” To mitigate this, SpaText [4] integrates a global text description for the overall scene with **local spatio-textual inputs**. This allows users to specify the precise details of certain objects or areas.

Instead of requiring dense segmentation maps, users can provide a sparse segmentation map which describes only key regions and leave the rest of the image to the model’s creativity. Furthermore, they can precisely control the layout, placement, and attributes of objects in the generated scene without requiring extensive expertise. For example, suppose we want to generate an image of ”a Labrador sitting near a blue ball on the beach.” Stable Diffusion might generate a Labrador and a ball, but their positions, sizes, or colors may not match the description. The beach might dominate the scene, making the objects secondary. On the other hand, **SpaText** places the Labrador in a specified location with the correct color and attributes. Furthermore, it maintains the overall context of the beach.

Besides the traditional global prompt $t_prompt$ that describes the entire scene, Spa-Text introduces the **spatio-textual matrix** $RST$ which is a structured input provided by the user to specify spatial and textual details for image generation. Each entry $RST[i, j]$ in the matrix either specifies the textual description of the content at pixel $[i, j]$ or is marked as $\emptyset$ (null) if no specific description is provided for that pixel. For instance, if the global description $t_{\text{global}}$ is *”a sunny day at the beach,”* the $RST$ matrix could specify Region A containing ”a brown dog sitting on the sand.”, Region B containing ”a red umbrella.” and other pixels as $\emptyset$.

The spatio-textual matrix $RST$ is created differently during the training and inference stages. The overall goal however in both cases is to align semantic embeddings with their spatial regions in the image. During the training stage, $RST$ is generated using a panoptic segmentation model to divide the input image $x$ into $N$ segments $\{S_1, S_2, \ldots, S_N\}$. Each segment $S_i$ corresponds to a distinct region or object in the image with a binary mask $M_i$ indicating the pixels belonging to $S_i$. To focus on meaningful regions, small segments that cover less than 5% of the image are excluded. From the remaining segments, $K$ segments are randomly selected for further processing to introduce variability and improve generalization during training.

For each selected segment $S_i$, the model crops a tight bounding box around the segment, masking out all other pixels to ensure that the embedding captures only the content of $S_i$. This cropped region is resized to match the input size required by the CLIP image encoder ($\text{CLIP}_{\text{img}}$). We then pass this to the CLIP encoder which generates a semantic embedding $\text{CLIP}_{\text{img}}(S_i)$ for the segment. The embeddings of these segments are then spatially mapped back into the spatio-textual matrix $ST_x$ according to their original positions in the image. For every pixel $(j, k)$ in the image, if it belongs to the segment $S_i$, the corresponding entry in $ST_x$ is set to $\text{CLIP}_{\text{img}}(S_i)$. Pixels outside the selected segments are filled with a zero vector $\vec{0}$. This can be expressed as:

$$ST_x[j,k] = \begin{cases} \text{CLIP}_{\text{img}}(S_i), & \text{if } (j,k) \in S_i, \\ \vec{0}, & \text{otherwise.} \end{cases} \tag{22}$$

This training stage helps RST understand how specific regions of the image relate to their semantic descriptions. For example, if a segment is labeled as "a brown dog sitting on grass," the training process helps the model associate the embedding for "brown dog" with the corresponding spatial region and its visual features.

During inference, $RST$ is created from user-provided inputs. The user supplies a global text prompt $t_{\text{global}}$, a sparse segmentation map specifying the regions of interest, and free-form textual descriptions for each region. For each region, the textual description $t_{\text{local}}$ is embedded using the CLIP text encoder ($\text{CLIP}_{\text{txt}}$). To ensure compatibility with the embeddings used during training, the text embeddings are transformed into the image embedding space using a prior model $P$. This transformation can be expressed as:

$$P(\text{CLIP}_{\text{txt}}(t_{\text{local}})) \to \text{CLIP}_{\text{img}}. \tag{23}$$

The transformed embeddings are then mapped back into $RST$ using the spatial information from the user-provided segmentation map. Each pixel in a specified region is assigned the embedding corresponding to that region's description. This process creates a sparse spatio-textual matrix where only the described regions are explicitly represented and leave the remaining areas to be inferred by the model.

In the context of latent diffusion models (such as Stable Diffusion), the spatio-textual matrix ($RST$) is incorporated into the generative process by conditioning the latent denoising steps with both global textual prompts and the localized information encoded in $RST$. Here's a detailed explanation of how $RST$ integrates into the latent diffusion model:

The $RST$ matrix provided by the user (or derived during training) is a spatial map aligned with the dimensions of the original image $x_0$. Since the diffusion process operates in the latent space, $RST$ is downsampled to match the spatial dimensions of the latent representation $z_t$. This ensures alignment between the latent variables and the spatial-textual conditioning.

At each diffusion step $t$, the noisy latent representation $z_t$ is concatenated with the downsampled $RST$ along the channel dimension. If $z_t$ has a channel depth $C$, and $RST$ has a channel depth $d_{\text{CLIP}}$, the combined input has a shape of:

$$(H, W, C + d_{\text{CLIP}}),$$

where $H$ and $W$ are the spatial dimensions of the latent space.

This concatenated input ensures that the model has access to both the latent image representation and the spatial-textual information encoded in $RST$. The latent denoising U-NeT $f$ takes the concatenated input and produces the next denoised latent $z_{t-1}$ based on the noise schedule $t$, the global text prompt $t_{\text{global}}$, and $RST$:

$$z_{t-1} = f(z_t, \text{CLIP}_{\text{txt}}(t_{\text{global}}), RST, t).$$

The model predicts the noise to be subtracted, ensuring that the output latent aligns with both the global context and the localized details specified by $RST$.

## 2.4 Prompt-to-Prompt

Prompt-to-Prompt [18] (P2P) was the first paper that explored **training-free** modification of Stable Diffusion II pipeline for edits. In SpaText, in order to introduce controllability we had to provide a semantic map that can control the generation of our diffusion model. This approach has provided appealing results but the masking procedure is cumbersome. Furthermore, editing capabilities such as modifying the texture of a specific object are out of the capabilities of T2I diffusion models using Spa-Text methodology.

The methodology of Prompt-to-Prompt leverages the cross-attention layers within text-conditioned diffusion models. As discussed, these layers map the relationship between text tokens and the spatial layout of the image. Let $\mathcal{J}$ by an image which was generated by a text-guided diffusion model using the text prompt $\mathcal{P}$. The goal is to edit the input image guided only be the edited prompt $\mathcal{P}^*$ resulting in an edited image $\mathcal{J}^*$. For example, consider an image generated from the prompt "my new bicycle", and assume that the user wants to edit the color of the bicycle, its material, or even replace it with a scooter while preserving the appearance and structure of the original image. An intuitive interface for the user is to directly change the text prompt by further describing the appearance of the bikes or replacing it with another word.

Each diffusion step t consists of predicting the noise $\epsilon$ from a noisy image $\vec{z}_t$ and text embedding $\phi(\mathcal{P})$ using U-net. At the final step, this process yields the generated image $\mathcal{J} = \vec{z}_0$. The interaction between the two modalities $\phi(\mathcal{P})$ and $\vec{z}_t$ occurs during the noise prediction, where the embeddings of the visual and textual features are fused using Cross-attention layers that produce spatial attention maps for each textual token.

The features of the noisy image $\vec{z}_t$ are projected to $\chi(\vec{z}_t)$. Then, a Query, Key and Value are calculated using cross-attention:

$$O_i = MV = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{24}$$

Where, $Q = W_Q \chi(\vec{z}_t)$, $K = W_k \phi(\mathcal{P})$ and $V = W_V \phi(\mathcal{P})$.

The attention matrix M is of size $NxM$ where N are the number of tokens. Since we have an N$-$ tokens from the Query and M tokens from the Key, the matrix $M_{ij}$ defines the weight of the value of the j -th token on the pixel (i,j).

The authors of the paper find that the spatial layout and geometry of the generated image depend on the cross-attention maps. Pixels are more attracted to the words that describe them, e.g., pixels of the bear are correlated with the word "bear". Note that averaging is done for visualization purposes, and attention maps are kept separate for each head in our method. Interestingly, we can see that the structure of the image is already determined in the early steps of the diffusion process.

**INSERT PICTURE HERE**

Since the attention reflects the overall composition, we can inject the attention maps M that were obtained from the generation with the original prompt $\mathcal{P}$ into a second generation with the modified prompt $\mathcal{P}^*$. This allows the synthesis of an edited image $\mathcal{I}^*$ that is not only manipulated according to the edited prompt, but also preserves the structure of the input image $\mathcal{J}$.

Let $\mathrm{DM}(\vec{z}_t, \mathcal{P}, \mathrm{t}, \mathrm{s})$ be the computation of a single step t of the diffusion process which outputs the noisy image $\vec{z}_{t-1}$ where ' $s$ ' is the random seed. The random seed $s$ is used to control the randomness introduced during the generation process and ensures that the process can be deterministic and reproducible if the same seed is used again.

Let V and M denote the value matrix and attention matrix produced by the original prompt $\mathcal{P}$. The authors denote $DM(\vec{z}_t, \mathcal{P}, \mathrm{t}, \mathrm{s})\{M \leftarrow \widehat{M}\}$ as the diffusion step where the attention map M is overridden by an additional given map $\widehat{M}$ but the values V are kept the same from the previous prompt. They denote $M_t^*$ as the produced attention map using the edited prompt $\mathcal{P}^*$. Lastly, they define Edit $(M_t, M_t^*, t)$ to be a general edit function, receiving as input the $t'$ th attention maps of the original and edited images during their generation. They then propose the following Promp-to-Prompt image editing map:

**Algorithm 1** Prompt-to-Prompt Image Editing

**Input:** A source prompt $\mathcal{P}$, a target prompt $\mathcal{P}^*$, and a random seed $s$.

**Output:** A source image $x_{\text{src}}$ and an edited image $x_{\text{dst}}$.

```
$z_T \sim \mathcal{N}(0, I)$                              ▷ Sample a unit Gaussian random variable with random seed $s$
$z_T^* \leftarrow z_T$
for $t = T, T-1, \ldots, 1$ do
    $z_{t-1}, M_t \leftarrow \text{DM}(z_t, \mathcal{P}, t, s)$                ▷ Denoise source prompt
    $M_t^* \leftarrow \text{DM}(z_t^*, \mathcal{P}^*, t, s)$                   ▷ Denoise target prompt
    $\widehat{M_t} \leftarrow \text{Edit}(M_t, M_t^*, t)$                    ▷ Perform editing between source and target maps
    $z_{t-1}^* \leftarrow \text{DM}(z_t^*, \mathcal{P}^*, t, s_t)\{M \leftarrow \widehat{M_t}\}$    ▷ Apply edited map to the target
end for
Return $(z_0, z_0^*)$
```

Let us understand the algorithm defined in detail. We begin with a source prompt $\mathcal{P}$ and the edited prompt $\mathcal{P}^*$ and the random seed s . At time T , we sample from a unit Gaussian: $\vec{z}_T \sim \mathcal{N}(0, \mathbb{I})$. For time t = T, to . . . 1, we now carry out the following:

- We run the diffusion processed conditioned on our original prompt $\mathcal{P}$. We denote this as $DM\left(\vec{z}_t, \mathcal{P}, t, s\right)$. This gives us $\vec{z}_{t-1}$ and $M_t$ where $M_t$ are the attention map associated with original diffusion process.

- We now copy the noise vector $\vec{z}_t$ and denote that as $\vec{z}_t^*$. We now run the diffusion process on the edited prompt $\mathcal{P}$ and the copy of the noise vector: $DM\left(\vec{z}_t^*, \mathcal{P}^*, t, s\right)$. This gives us the new modified attention map $M_t^*$.

- We now perform Editing, in which we take the modified attention map $M_t^*$ and the original attention map $M_t$. This yields a new map $\widehat{M_t}$. To sum up, $\widehat{M_t} \leftarrow \text{Edit}\left(M_t, M_t^*, t\right)$.

- Using the edited map $\widehat{M_t}$, the diffusion model predicts the next latent variable $\vec{z}_{t-1}^*$ which is denoted as $\vec{z}_{t-1}^* \leftarrow DM\left(\vec{z}_t^*, \mathcal{P}^*, t, s\right)\left\{M \leftarrow \widehat{M_t}\right\}$

**Defining Editing Functions**

The main crux is how do we go about defining $\widehat{M_t} \leftarrow \text{Edit}\left(M_t, M_t^*, t\right)$. The following are the three key Edits defined.

1. **Word Swap**

2. **Adding a new phrase**

3. **Attention Re-weighting**

**Word Swap**: Suppose the user swaps the tokens of the original prompt with others. For example, $\mathcal{P}$ = ”a big red bicycle” to $\mathcal{P}^*$ = ”a big red car”. Now, if we overly constraint the generation of the new image with the attention map $M$ of the original prompt $\mathcal{P}$, the new content might not be fully realized (e.g., the ”car” might still look like a ”bicycle”).

To address this, the authors utilize attention injection - a process of controlling the amount and timing of attention map injection during the diffusion process. To balance between preserving the original structure and allowing the new content to appear correctly, the process uses a timestamp parameter ($\tau$). Up the to the timestep $\tau$, the modified attention map $M_t^*$ is used. This allows the new object (car) to start forming based on the new prompt. After $\tau$, the original attention map $M_t$ from the source image is reintroduced, helping to maintain the composition and structure of the original image. Formally, we express this as following:

$$\text{Edit}\left(M_t, M_t^*, t\right) = \begin{cases} M_t^*, & \text{if } t < \tau \\ M_t, & \text{otherwise} \end{cases} \tag{25}$$

This means that early in the diffusion process, the new content has more freedom to form using $M_t^*$ but as the process progresses, the original structure is reinforced by switching back to $M_t$. If $\tau$ is set high (near the end of the diffusion process), the final image will look more like the original image but with the new object in place (e.g., a car that might still resemble the original bicycle's shape). If $\tau$ is set low (near the beginning of the process), the new content (car) will have more freedom to adopt its correct shape and appearance, potentially altering the structure more significantly.

**Adding a New Phrase**: Suppose the user adds new tokens to the prompt, e.g., $\mathcal{P}$ = "a castle next to a river" to $\mathcal{P}^*$ = "children drawing of a castle next to a river". When you add new tokens (words or phrases) to a prompt (e.g., changing "a castle next to a river" to "children drawing of a castle next to a river"), the generated image needs to reflect the new stylistic or contextual changes (e.g., the drawing style), while still retaining the original elements that are common between both prompts (e.g., the castle and river).

The attention alignment function A is used to map tokens from the modified prompt $\mathcal{P}^*$ back the original prompt $\mathcal{P}$. For each token in the modified prompt $\mathcal{P}^*$, the function A identifies whether this token matches a token in the original prompt $\mathcal{P}$. If it does, A returns the corresponding index from P ; it doesn't A , return none.

The attention injection is applied selectively based on the alignment function $A$. Formally,

$$\text{Edit}\left(M_t, M_t^*, t\right)_{i,j} = \begin{cases} \left(M_t^*\right)_{i,j}, & \text{if } A(j) = \text{None} \\ \left(M_t\right)_{i,A(j)}, & \text{otherwise} \end{cases} \tag{26}$$

Here, i corresponds to a pixel in the image, and j corresponds to a token in the text prompt.
To understand this , take $\mathcal{P}$ = "a castle next to a river" and $\mathcal{P}$ * = "children drawing of a castle next to a river". The tokens for these are, respectively:

$$\mathcal{P} = [\text{"a", "castle", "next", "to", "a", "river"}]$$
$$\mathcal{P}^* = [\text{"children", "drawing", "of", "a", "castle", "next", "to", "a", "river"}]$$

For "children", "drawing", "and", "of", $A(j) = None$. For "a," "castle," "next," "to," and "river," A(j) maps to the corresponding indices in the original prompt $\mathcal{P}$. For new tokens ("children," "drawing," "of"), the attention map $M_t^*$ is used. For common tokens ("castle," "river"), the attention map $M_t$ from the original prompt is used.

**Attention Re-weighting.**: Lastly, the user may wish to strengthen or weakens the extent to which each token is affecting the resulting image. For example, consider the prompt P = "a fluffy red ball", and assume we want to make the ball more or less fluffy. To achieve such manipulation, we scale the attention map of the assigned token j∗ with parameter c $\in [-2, 2]$, resulting in a stronger/weaker effect. The rest of the attention maps remain unchanged. That is:

$$\text{Edit}\left(M_t, M_t^*, t\right)_{i,j} = \begin{cases} c\left(M_t\right)_{i,j}, & \text{if } j = j^* \\ \left(M_t\right)_{i,j}, & \text{otherwise} \end{cases} \tag{27}$$

## 2.5 Plug and Play

The methodology [31] builds upon P2P framework. P2P's manipulation of cross-attention maps allows for control at the object level (i.e., associating large regions of the image with specific words). However, it may not be able to preserve more localized spatial details, such as the parts of an object (e.g., the legs of a chair or the wings of a butterfly). This is because the cross-attention maps primarily capture broad associations between words and spatial regions, not the fine-grained details of those regions. Since cross-attention maps are formed by associating spatial features with words, they might capture the overall region corresponding to an object but miss out on more detailed spatial information that isn't explicitly mentioned in the text prompt. For example, if the text describes ”a car,” P2P might accurately position the car in the image but could struggle to precisely place the wheels or headlights if those parts aren't specifically mentioned in the prompt.

The paper addresses the task of transforming an image into another image based on a textual description. For example, transforming a photo of a house into a version that looks like it was drawn by a child, based on the prompt ”children's drawing of a house.”. Formally, given an input guidance image $I^G$ and a target prompt P , the goal of the paper is to generate a new image $I^*$ that complies with P and preserves the structure and semantic layout of $I^G$. The paper utilizes StableDiffusion to carry this out.

Since the work utilizes intermediate features, it is a good idea to formalize what they are. We know that denoising nets use U-Net. Layers of the U-Net comprise a residual block, a selfattention block, and a cross-attention block. The residual block convolve image features $\phi_t^{l-1}$ from the previous layer $l-1$ to produce intermediate features $\vec{f}_t^l$. In the self-attention block, features are projected into queries $\vec{q}_t^l$, keys, $\vec{k}_t^l$ and values $\vec{v}_t^l$ and the output block is given by:

$$\hat{f}_t^l = A_t^l \vec{v}_t^l \text{ where } A_t^l = \text{Softmax}\left(\vec{q}_t \vec{k}_t^{l^T}\right) \tag{28}$$

Consider the guidance image $I^G$. We pass it through the VQ-VAE to obtain the latent $\vec{z}_0$. Instead of noising it to obtain the noisy latent $\vec{z}_T$ as we do in Img-to-Img, we perform DDIM inversion in which a U-NeT is used to estimate the noisy latent $\vec{z}_T$. Thus, we obtain:

$$\vec{z}_T^G = DDIM - \text{Inv}\left(I^G\right) \tag{29}$$

Where $\vec{z}_T^G$ is the noise obtained by inverting the guidance image $I^G$. This noise serves as the starting point for generating a new image $I^*$ based on a different prompt $P$.

**Insert figure here**

With the prompt P being, ”a photo of a golden robot” which we want in the style of the image $I^G$. We tokenize our prompt and create a vector embedding $\tau^\theta(\vec{c})$ on which the denoising process will be carried out. Running the reverse diffusion process, we have:

$$\vec{z}_{t-1}^G = \varepsilon_\theta\left(\vec{x}_T^G, \emptyset, t\right) \tag{30}$$

This process extracts the guidance features $\left\{f_t^l\right\}$ where $l$ denotes the layer of the model. These features capture the spatial and semantic information of the image as it is being refined from noise. For example, layer 1 = 1 will be the downsampling ResNet block all the way to layer 7 which would be the upsampling ResNet block.

Now, in a typical denoising process, the model generates a new set of features $\left\{f_t^{l*}\right\}$ based on the noisy image $z_t^*$. What we do in this case, is that we inject the features $\left\{f_t^l\right\}$ from the guidance image into the denoising steps of $z_t^*$, thereby overriding $\left\{f_t^{l*}\right\}$. This operation can be expressed as:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, \tau^\theta(\vec{c}), t; \left\{f_t^l\right\}\right) \tag{31}$$

On the other hand, in case of no injection, we have:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t; \emptyset\right) = \varepsilon_\theta\left(\vec{x}_t^*, \tau^\theta(\vec{c}), t\right) \tag{32}$$

The figure below shows the effect of injecting features $\left\{f_t^l\right\}$ :

**INSERT FIGURE HERE**

As seen, injecting features only at layer $l = 4$ is insufficient for preserving the structure of the guidance image. As we inject features in deeper layers, the structure is better preserved, yet appearance information is leaked into the generated image (e.g., shades of the red tshirt and blue jeans are apparent in Layer 4-11). To achieve a better balance between preserving the structure of $I^G$ and deviating from its appearance, we do not modify spatial features at deep layers, but rather leverage the self-attention layers.

The authors consider the attention matrices $A_t^{l^*}$ to achieve fine-grained control over the generated control. The figure below shows the leading principal components $A_t^{l^*}$ for a given image:

**INSERT FIGURE HERE**

As seen, in early layers, the attention is aligned with the semantic layout of the image, grouping regions according to semantic parts. Gradually, higher-frequency information is captured. Practically, injecting the self-attention matrix is done by replacing the matrix $A_t^{l^*}$ by the modified attention matrix $A_t^l$. Intuitively, this operation pulls features close together, according to the affinities encoded in $A_t^l$. We denote this by process:

$$\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t; \quad f_t^4, \left\{A_t^l\right\}\right) \tag{33}$$

The figure below shows the effect of this attention-injection:

**Insert figure here**

As seen, with only self-attention, i.e., $\vec{z}_{t-1}^* = \varepsilon_\theta\left(\vec{x}_t^*, P, t, \left\{A_t^l\right\}\right)$, there is no semantic association between the original content and the translated one, resulting in large deviations in structure.

The plug-and-play diffusion features framework is summarized in Alg. 1 below, and is controlled by two parameters: (i) $\tau_f$ defines the sampling step t until which $f_t^4$ are injected. (ii) $\tau_A$ is the sampling step until which $A_t^l$ are injected. In all our results, we use a default setting where selfattention is injected into all the decoder layers.

**Algorithm 2** Plug-and-Play Diffusion Features

```
Inputs: $I^G$: real guidance image, $P$: target text prompt, $\tau_f, \tau_A$: injection thresholds
Initialization:
    • $\boldsymbol{x}_T^G \leftarrow \text{DDIM-inv}(I^G)$                ▷ Invert guidance image into latent space
    • $\boldsymbol{x}_T^* \leftarrow \boldsymbol{x}_T^G$                ▷ Start from the same seed
for $t = T, T-1, \ldots, 1$ do
    $\boldsymbol{z}_{t-1}^G, \boldsymbol{f}_t^4, \{\boldsymbol{A}_t^l\} \leftarrow \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t^G, \varnothing, t)$
    $\boldsymbol{x}_{t-1}^G \leftarrow \text{DDIM-samp}(\boldsymbol{x}_t^G, \boldsymbol{z}_{t-1}^G)$
    if $t > \tau_f$ then
        $\boldsymbol{f}_t^{*4} \leftarrow \boldsymbol{f}_t^4$
    else
        $\boldsymbol{f}_t^{*4} \leftarrow \varnothing$
    end if
    if $t > \tau_A$ then
        $\boldsymbol{A}_t^{*l} \leftarrow \boldsymbol{A}_t^l$
    else
        $\boldsymbol{A}_t^{*l} \leftarrow \varnothing$
    end if
    $\boldsymbol{z}_{t-1}^* \leftarrow \hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_t^*, P, t; \boldsymbol{f}_t^{*4}, \{\boldsymbol{A}_t^{*l}\})$
    $\boldsymbol{x}_{t-1}^* \leftarrow \text{DDIM-samp}(\boldsymbol{x}_t^*, \boldsymbol{z}_{t-1}^*)$
end for
    Output: $I^* \leftarrow \boldsymbol{x}_0^*$
```

## 2.6 Attend and Excite

The work Attend and Excite [9] addresses the problem of **"catastrophic neglect"** where the model fails to generate one or more of the subjects from the input prompt. Moreover, the authors note that in some cases the model also fails to correctly bind attributes (e.g., colors) to their corresponding subjects. They label this problem as **"attribute binding"**. The figure below shows an example of both of these cases:

**insert figure here**

To help mitigate these problems, the authors introduce an attention-based formulation dubbed Attend-and-Excite. This guides the model to refine the cross-attention units to attend to all subject tokens in the text prompt and strengthen - or excite - their activations, encouraging the model to generate all subjects described in the text prompt. In order for a subject to be present in the generated image, the model should assign at least one image patch to the subject's token. Attend-and-Excite embodies this intuition by demanding that each subject token is dominant in some patch in the image.

The authors employ their method over Stable Diffusion Model. Let N be the number of text tokens in the prompt. If ( P × P ) denotes the number of patches in self-attention, then $P \in \{64, 32, 16, 8\}$ across the layer of Stable Diffusion II. An attention map $A_t \in \mathbb{R}^{P \times P \times N}$ is calculated over linear projections of the intermediate features $(Q)$ and text embedding $(K)$.

$A_t$ defines a distribution over the text tokens for each spatial patch $(i, j)$. Specifically, $A_t[i, j, n]$ denotes the probability assigned to token n for the ( i, j )-th spatial patch of the intermediate feature map. Intuitively, this probability indicates the amount of information that will be passed from token $n$ to patch $(i, j)$. Note that the maximum value of each of the $P \times P$ cells is 1 . The authors operate over ( $16 \times 16$ ) attention units since they have been shown to contain the most semantic information.

Intuitively, for a subject to be present in the synthesized image, it should have a high influence on some patch in the image. As such, the authors define a loss objective that attempts to maximize the attention values for each subject token. They then update the noised latent at time $t$ according to the gradient of the computed loss. This encourages the latent at the next timestep to better incorporate all subject tokens in its representation. This manipulation occurs on the fly during inference (i.e., no additional training is performed)

**The proposed algorithm:**

Let $\mathcal{P}$ be a text prompt that guides the image generation process and $\mathcal{S}$ be the Subject token indices which are extracted from the text prompt. These indices $\mathcal{S}$ identify the specific parts of the text that should be emphasized in the image generation process. Use the stable diffusion model SD to compute the attention map $A_t$ at the current timestep t given prompt $\mathcal{P}$ and $\vec{z}_t$.

Now, the authors note that Stable Diffusion learns to consistently assign a high attention value to the $\langle sot \rangle$ (start of text) token in the token distribution defined in $A_t$. Since we are interested in enhancing the actual prompt tokens, they re-weigh the attention values by ignoring the attention of $\langle sot \rangle$ and performing a Softmax operation on the remaining tokens. After the Softmax operation, the $(i, j)$-th entry of the resulting matrix $A_t$ indicates the probability of each of the textual tokens being present in the corresponding image patch.

For each subject, they calculate the corresponding attention values $A_t^S$. Now, the authors note that these may not fully reflect whether an object is generated in the resulting image.Specifically, a single patch with a high attention value could stem from partial information being passed from the token $s$. This may occur when the model does not generate the full subject, but rather a patch that resembles some part of the subject, e.g., a silhouette that resembles an animal's body part.

To avoid such solutions, the authors apply a Gaussian filter over $A_t^s$. After doing so, the attention value of the maximally activated patch is dependent on its neighboring patches since, after this operation, each patch is a linear combination of its neighboring patches in the original map.To understand how this helps, assume we have a $(4 \times 4)$ attention map $A_t^S$ :

$$A_t^S = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.1 \\ 0.4 & 0.8 & 0.6 & 0.3 \\ 0.2 & 0.5 & 0.9 & 0.4 \\ 0.1 & 0.2 & 0.3 & 0.2 \end{pmatrix}$$

A Gaussian filter is a type of linear filter that applies a Gaussian function to the values in a matrix, giving more weight to the central values and less weight to the distant ones:

$$G = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

To apply G, you perform a convolution of the attention map $A_t^S$ with the Gaussian filter G. Each element in the resulting smoothed map is a weighted sum of its neighboring elements. This "distributes" the attention weights and ensures a smoother, more robust representation.

Intuitively, successfully generated subjects should have an image patch that significantly attends to their corresponding token. For each subject token in $S$, the optimization loss introduces encourages the existence of at least one patch of $A_t^S$ with a high activation value. Therefore, they define the loss quantifying this desired behavior as:

$$\mathcal{L} = \max_{s \in S} \mathcal{L}_s \text{ where } \mathcal{L}_s = 1 - \max\left(A_t^S\right) \tag{34}$$

**Iterative Latent Refinement**. A single latent update has been made at each denoising timestep so far. However, if the attention values of a token do not reach a certain threshold in the early denoising stages, the corresponding object will not be generated. To address this issue, $z_t$ is iteratively updated until a predefined minimum attention value is achieved for all subject tokens. However, performing too many updates of $z_t$ may cause the latent to become out-of-distribution, resulting in incoherent images. Therefore, this refinement is performed gradually across a small subset of timesteps.

Specifically, each subject token is required to reach a maximum attention value of at least 0.8. To achieve this gradually, iterative updates are performed at various denoising steps. The iterations are set at $t_1 = 0$, $t_2 = 10$, and $t_3 = 20$ with minimum required attention values of $T_1 = 0.05$, $T_2 = 0.5$, and $T_3 = 0.8$. This gradual refinement helps prevent $z_t$ from becoming out-of-distribution while encouraging more faithful generations.

## 2.7 Structure Diffusion:

In the work "Training-Free Structured Diffusion Guidance for Compositional Text-To-Image Synthesis" [12] W Feng et al, the authors confront the problem of **attribute binding** and **constitutionality**.

For this, they utilize the concept of a parsing tree. These trees help identify relationships between different parts of a sentence, such as which adjectives (attributes) describe which nouns (objects). For example, in the sentence "a red apple," "red" is an attribute and "apple" is the object. The idea here is to use this structured information to guide the image generation process. Similarly, Structured representations like constituency trees or scene graphs represent the relationships between different elements in a sentence. A scene graph is another structured representation that maps objects and their relationships in a scene.

The authors of the paper utilize cross-attention-maps to build what they call **"structured cross-attention guidance"** which utilizes language parsers. A language parser is a tool that breaks down sentences into their grammatical components, revealing the hierarchical structure of the language. For example, in the sentence "The cat on the mat is sleeping," the parser would identify "The cat" as a noun phrase, "on the mat" as a prepositional phrase, and "is sleeping" as a verb phrase. These language parsers are used to create Hierarchical structures. These are the layers of relationships between different components in the text. For instance, "cat" might be part of a larger phrase "The cat on the mat," which itself is part of the complete sentence.

Similarly the prompt $P$: **"A blue chair and a red table in a green room."** involves three distinct objects and attributes: A **blue chair**, A **red table** and A **green room**. In the first stage of structure diffusion, **Constituency Trees** are employed to breaks down the sentence into hierarchical noun phrases (NPs): NP1: "blue chair", "red table", NP3: "green room". These phrases represent the key concepts $C = \{c_1, c_2, c_3\}$ in the prompt.

Next, a scene graph explicitly represents the relationships between objects.

- **Nodes:** Chair, Table, Room
- **Attributes:** Blue (chair), Red (table), Green (room)
- **Relations:** "Chair in room," "Table in room."

This structured representation provides a clear mapping of objects, attributes, and their relationships. The structured guidance modifies the cross-attention mechanism during the image generation process to ensure all concepts are faithfully represented.

The attention map $M_t$ is computed for the full prompt $P$ using the key $K_p$, which represents the entire prompt's text embedding. Separate value tensors $V = [V_p, V_1, V_2, V_3]$ are generated where $V_p$ represents the overall prompt and $V_1, V_2, V_3$ represents individual concepts (”blue chair,” ”red table,” ”green room”).

The output $O_t$ at each timestep is a weighted combination of the attention map $M_t$ and the value tensors $V_i$:

$$O_t = \frac{1}{k+1} \sum_{i=0}^{k} M_t V_i, \tag{35}$$

where $k$ is the number of concepts. This ensures that each concept $c_i$ (e.g., ”blue chair”) has a dedicated attention focus and the generated image explicitly includes all objects and attributes.

The method uses a loss function to enforce correct attribute binding and compositionality:

$$L = \sum_{c \in C} \left(1 - \max_{(i,j)} M_t[c]\right), \tag{36}$$

where $M_t[c]$ is the Attention map for concept $c$ (e.g., ”blue chair”) and $\max_{(i,j)} M_t[c]$ ensures that at least one patch in the image strongly attends to the concept.

For our example, the loss would penalize the model if no region in the image attends strongly to ”blue chair.” Similarly, the loss is computed for ”red table” and ”green room.”. Overall, the loss ensures all objects (chair, table, room) are included. Furthermore, the attributes (blue, red, green) are correctly assigned.

## 2.8 DiffEdit

The work [11] elegantly utilizes DDIM inversion to perform semantic edits. These are edits in which the edited image $I^*$ retains much of the features of the original image $I$. Given an image $I$, DiffEdit automatically generates a mask highlighting regions of the input image that need to be edited. Recall that we can modify the existing image to obtain its latent representation by using DDIM inversion:

$$\vec{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}} \vec{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1}} \vec{\epsilon}_\theta^{(t)} \tag{37}$$

In terms of the Neural ODE, we can express this formalism as following:

$$d\vec{u} = d\tau(t) \vec{\epsilon}_\theta^{(t)} \left( \frac{\vec{u}}{\sqrt{1+\tau^2}}, t \right) \tag{38}$$

The authors of the paper use equation 38 as their starting point. They parametrize the timestep $t$ to be between 0 and 1 , so that t = 1 corresponds to T steps of diffusion in the original formulation. They explain that, as proposed by Song et al. (2021), the ODE can be utilized to encode an image $\vec{x}_0$ into a latent variable $\vec{x}_r$ for a timestep $r \leq 1$, using the boundary condition $\vec{u}(0) = \vec{x}_0$ instead of $\vec{u}(t = 1)$. This encoding process is achieved by applying an Euler scheme up to timestep $r$. Throughout the paper, this process is referred to as DDIM encoding, with the corresponding function mapping $\vec{x}_0$ to $\vec{x}_r$ denoted as $E_r$, and $r$ referred to as the encoding ratio.

The authors highlight that, with sufficiently small steps in the Euler scheme, decoding $\vec{x}_r$ approximately reconstructs the original image $\vec{x}_0$. This property is particularly significant in the context of image editing as all the information of the input image $\vec{x}_0$ is encapsulated in $\vec{x}_r$ and can be retrieved through DDIM sampling.

Given the above is preamble, let us consider their framework more closely. The goal is to change specific parts of an image according to a text query while keeping the rest of the image unchanged. This is achieved by inferring a mask that identifies the region needing changes and then guiding the denoising process in the diffusion model.

A text-conditioned diffusion model is used to generate noise estimates for the image. This is done twice: once with a reference text (e.g., "horse") and once with the editing text query (e.g., "zebra"). More formally, let us denote the image by $\vec{x}_0.Q_{ref} =$ "horse" is the original prompt and editing text $Q =$ "horse" is the editing prompt. Using equation (38), we perform DDIM encoding to obtain noise estimates $\bar{\epsilon}_{ref}^{(t)}$ and $\bar{\epsilon}_{Q}^{(t)}$ respectively. We then calculate the difference between these two noise estimates to identity which regions of the image are affected by the text query:

$$\Delta\epsilon = |\epsilon_{ref} - \epsilon_Q| \tag{39}$$

To make the difference map more robust, Gaussian noise is added with a strength of 50%. Extreme values in the noise predictions are removed, and the differences are averaged over multiple noise samples (e.g., n = 10 ).

This can be express algorithmically as follows: For each sample i from 1 to n, 1. Add Gaussian noise to the image and then calculate $\Delta\epsilon_i = \left|\epsilon_{ref}^{(i)} - \epsilon_Q^{(i)}\right|$. After this, average the results $\Delta\epsilon_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n}\Delta\epsilon_i$ . Finally, rescale the averaged difference map $\Delta\epsilon_{\text{avg}}$ to the range $[0, 1]$ and binarize it using a threshold (e.g 0.5) :

$$M(x, y) = \begin{cases} 1 \text{ if } \Delta\epsilon_{\text{avg}}(x, y) > 0.5 \\ 0, \text{ otherwise} \end{cases} \tag{40}$$

Finally, encode the input image $\vec{x}_0$ using DDIM encoding function $E_r$ at timestep r without any text conditioning. Decode the latent $\vec{x}_r$ back into an image using the diffusion model, conditioned on the editing text query Q . During the denoising process, use the mask M to ensure that only the masked regions of the image are edited.

## 2.9 Dense T2I Generation

The work "Dense Text-to-Image Generation with Attention Modulation" [22] provides users with controllability for prompts which contain a large number of concepts. To begin with, it breaks down the prompt $\mathcal{P}$ into a set of non-overlapping segments. For example, consider the prompt $\mathcal{P} =$ "A painting of a couple holding a yellow umbrella ($\vec{c}_1$) in a street on a rainy night. The woman is wearing a white dress ($\vec{c}_2$) and the man is wearing a blue suit ($\vec{c}_3$).

It then aims to map each non-overlapping caption $\vec{c}_i$ with an associated binary map $\vec{m}_i$ where 1 in the binary map indicates the presence of a feature in a specific area of the image and 0 indicates that the feature is not present in the image. The schematic of this for our caption is shown in the figure below:

**INSERT FIGURE HERE**

Formally, the condition is defined as a set of N segments $\{(\vec{c}_n, \vec{m}_n)\}_{n=1}^{N}$ where each segment:

$$(\vec{c}_n, \vec{m}_n) \tag{41}$$

In equation 41, it is important to note that the dimensionality of $\vec{m}_n$ will be determined by patches. For example, suppose we have an image $\mathcal{J} \in \mathbb{R}^{HxWx3}$ and we patchify this into ( $\mathrm{P}, \mathrm{P}$ ). Then $\vec{m}_n$ would also be a 2D matrix of size (P, P). For example, if an image is converted into $(16, 16)$ patches, then a binary $\vec{m}_n$ associated with an object placed on the left of the image might have 1 's there and 0 everywhere else.

Given the input conditions, the aim is to modulate attention maps of all attention layers so that the object described by $\vec{c}_n$ can be generated in the corresponding region $\vec{m}_n$. In order to modulate the attention for N segments consisting of $(\vec{c}_n, \vec{m}_n)$, the authors introduce a **condition map $\boldsymbol{R}$**. The condition map $\boldsymbol{R}$ defines whether to increase or decrease the attention score a particular pair. If two tokens belong to the same segment, they form a positive pair, and their attention score will be increased. If not, they form a negative pair, with their attention decreased.

We will consider the particular form of $\mathbf{R}$ for both cross-attention and self-attention.Firstly, lets begin with cross-attention. Let the condition $\vec{y}$ be converted into N tokens and the image be converted into a ( $\mathrm{P} \times \mathrm{P}$ ) patch. Then, an attention map $A_t^S \in \mathbb{R}^{P \times P \times N}$ is calculated over linear projections of the intermediate features $(Q)$ and text embedding $(K)$

Assume now that we have N segments $\{(\vec{c}_n, \vec{m}_n)\}_{n=1}^{N}$ where $\vec{m}_n = \{0,1\}^{P^2} \in \mathbb{R}^{PxP}$. We flatten this map into a vector of $P^2$-dimensions. We now define a function $\tilde{k}[j]$ which maps the j -th key token (remember that the key tokens are produced from text) to a segment index n. For example, if the j -th key token corresponds to a patch that belongs to the yellow umbrella, $\tilde{k}[j]$ will map j to 1. Suppose that another key token $\left(7^{\text{th}}\right)$ belongs to the blue car, then $\tilde{k}[2]$ will be mapped to 2 .

The authors then introduce the following form of $\mathbf{R}$ :

$$R_{:j}^{\text{cross}} = \begin{cases} 0 \text{ if } \vec{k}[j] = 0 \\ \vec{m}_{\vec{k}[j]} \text{ otherwise} \end{cases} \tag{42}$$

$R_{:j}^{cross}$ represents the modulated attention map for the j-th text token.
Let us take a toy example to understand this. Suppose we have a spatial attention map of dimensions (4 x 4):

$$A_t = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

The associated Prompt is:
["A", "painting", "of", "a","couple", "holding", "a", "yellow", "umbrella", "The", "woman", "is", "wearing", "a", "white", "dress"]

The 2 Segments are:

- $(\vec{c}_1, \vec{m}_1) = \vec{c}_1 = (\text{ a yellow umbrella }), \vec{m}_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

- $(\vec{c}_2, \vec{m}_2) = \vec{c}_2 =$ (the woman is wearing a white dress), $\vec{m}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

Suppose the token is "a", which is the first key. Since that token does not correspond to any segment, we have $\vec{k}[a] = \vec{k}[1] = 0, \vec{k}[\text{ painting }] = \vec{k}[2] = \overrightarrow{0}, \vec{k}[of] = \vec{k}[3] = \overrightarrow{0}, \vec{k}[a] = \vec{k}[4] = \overrightarrow{0}$ and so on where $\overrightarrow{0} \in \mathbb{R}^{4\times4}$. On the other hand, the token $\vec{k}[\text{ yellow }] = \vec{k}[7], \vec{k}[umbrella] = \vec{k}$ [8] does match with $\vec{c}_1$. Hence, for these key tokens, the cross-attention matrix $R^{\text{cross }}$ is mapped to $\vec{m}_1$.

$$R^{\text{cross }} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Similarly, for the tokens, "the", "woman", "is", "wearing", "a", "white", "dress", the crossattention matrix is mapped to $\vec{m}_2$ :

$$R^{\text{cross }} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

**Defining** $M_{\mathbf{pos}}$ , $M_{\mathbf{neg}}$ , $S$ **:** Now that we have defined $R^{\text{cross }}$ which allows us to selectively associate a text token with a specific region of interest in the image, we would like more control over the values of R (Right now, the value of $R^{\text{cross }}$ is 1 for our region of interest). For this, we can define $M_{pos}$ which enhance the attention scores between query and key tokens that belong to the same segment. For example, if want to enhance the region associated with same segment by 0.7 , we would have:

$$\mathrm{M} = R \odot M_{pos} \tag{43}$$

Where for our example for the tokens "the", "woman", "is", "wearing", "a", "white", "dress":

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 0.7 & 0.7 & 0.7 & 0.7 \\ 0.7 & 0.7 & 0.7 & 0.7 \\ 0.7 & 0.7 & 0.7 & 0.7 \\ 0.7 & 0.7 & 0.7 & 0.7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0.7 \\ 0 & 0 & 0.7 & 0.7 \end{bmatrix}$$

We can incorporate even more control. Suppose we want to reduce the attention values for tokens not in the same segment for a particular text token. Then,

$$M = R \odot M_{pos} - (\mathbb{I} - R) \odot M_{neg} \rightarrow (4)$$

For our example, this would be equal to (for $M_{\text{pos }} = 0.7$ and $M_{\text{neg }} = 0.2$ ):

$$M = \begin{bmatrix} -0.2 & -0.2 & -0.2 & -0.2 \\ -0.2 & -0.2 & -0.2 & -0.2 \\ -0.2 & -0.2 & 0.7 & 0.7 \\ -0.2 & -0.2 & 0.7 & 0.7 \end{bmatrix}$$

The authors further incorporate a matrix S which takes into account the area that each segment (or object) occupies in the image. Larger segments (e.g., a large car in the image) would result in higher values in the corresponding areas of S , while smaller segments (e.g., a small umbrella) would result in lower values. In our example, both the segments are equally sized. Out of 16 patches, they occupy 4 patches so $S_{ij} = 0.25$ for both of them. With S, equation (4) is modified as following:

$$M = R \odot M_{pos} \odot (1 - S) - (\mathbb{I} - R) \odot M_{neg} \odot (1 - S) \rightarrow (5)$$

For our example, this would give:

$$M = \begin{bmatrix} -0.45 & -0.45 & -0.45 & -0.45 \\ -0.45 & -0.45 & -0.45 & -0.45 \\ -0.45 & -0.45 & 0.95 & 0.95 \\ -0.45 & -0.45 & 0.95 & 0.95 \end{bmatrix}$$

Finally, the authors observe that a large modification may deteriorate the image quality as the timestep $t$ approaches zero. Therefore, they use a scalar $\lambda_t$ to adjust the degree of modulation using the power function as below:

$$\lambda_t = wt^p \rightarrow (6)$$

Where t $\in \boldsymbol{w}$.t$^p$.

Keeping all of this in mind, Modulation matrix M becomes:

$$M = \lambda_t \cdot R \odot M_{pos} \odot (1 - S) - \lambda_t \cdot (1 - R) \odot M_{neg} \odot (1 - S) \rightarrow \qquad (6)$$

Consider a general Query, Key and Value matrix calculated in attention:

$$O_i = AV = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

With the introduction of above paradigm, the authors of the paper modify $A$ with a modulated attention map:

$$A' = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d}}\right) \rightarrow (7)$$

**Modulating self-attention maps:**: Since the modulation matrix R for self-attention is straightforward, I did not say much on it. Here we describe it. Recall that self-attention considers how the i-th patch correlates with the j-th patch (there is no conditioning). The authors propose a modulation designed to restrict communication between tokens of different segments, thereby preventing the mixing of features of distinct objects. Specifically, they increase the attention scores for tokens in the same segment and decrease it for those in different segments. The query-key pair condition map $R_{\text{self}}$ for this objective is defined as:

$$R_{ij}^{\text{self}} = \begin{cases} 1 \text{ if } \exists \mathrm{ns.t} \overrightarrow{\mathrm{m}}_{\mathrm{n}}[\mathrm{i}] = 1 \text{ and } \overrightarrow{\mathrm{m}}_{\mathrm{n}}[\mathrm{j}] = 1 \\ 0, \text{ otherwise} \end{cases} \tag{44}$$

token on the image features is modulated according to the layout condition.

## 2.10 LLM-Diffusion

The authors of [24] equip the diffusion models with an LLM that provides grounding for enhanced prompt understanding. Firstly, they train an LLM using incontext-learning. This is an illustration of prompt-engineering. Under this paradigm, the model is given a prompt that includes a task description, along with a few examples (in some cases, known as "few-shot learning"). These examples serve as a guide for the model to understand the task. .

To generate the layout of an image, their method embeds the input text prompt $\vec{y}$ into a template and queries an LLM for completion. For example, suppose we given the following prompt: $\mathcal{P}$ = "A realistic photo of a gray cat and an orange dog on the grass". They embed the prompt into the following in context template for LLM: "Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a background prompt describing the scene..." together with In-context examples. An LLM processes this to give an output that may appear as following:

**Caption:** A realistic photo of a gray cat and an orange dog on the grass.

**Objects:** [

$('agraycat', [50, 120, 180, 200]),$

$('anorangedog', [300, 120, 180, 200]),$

$('grass', [0, 340, 512, 172])$ ]

**Background prompt:** A realistic photo of a grassy outdoor scene.

**Negative prompt:**

It will comprise of two components: 1) a captioned bounding box for each foreground object, with coordinates specified in the ( $\mathrm{x}, \mathrm{y}$, width, height) format, and 2 ) a simple and concise caption describing the image background along with an optional negative prompt indicating what should not appear in a generated image. The negative prompt is an empty string when the layout does not impose restrictions on what should not appear.

The resulting layout from the LLM completion is then parsed and used for the subsequent image generation process. For each foreground object i in the image layout, the authors first generate an image with a single instance by denoising from $\vec{z}_T^{(i)}$ to $\vec{z}_0^{(i)}$ where $\vec{z}_T^{(i)}$ refers to the latents of object i at denoising timestep $t$. In this denoising process, we use "[background prompt] with [box caption]" (e.g., "a realistic image of an indoor scene with a gray cat") as the text prompt for denoising. The initial noise latent is shared for all boxes to ensure globally coherent viewpoint, style, and lighting.

To ensure the object aligns with the bounding box, we manipulate the cross-attention maps $A^{(i)}$ of the noise-prediction network. Each map describes the affinity from pixels to text tokens:

$$A_{uv}^{i} = \operatorname{softmax}\left(\vec{q}_u^T \vec{k}_v\right) \tag{45}$$

Where $\vec{q}_u$ and $\vec{k}_v$ are linearly transformed image feature at spatial location $u$ and text feature at token index v in the prompt, respectively

Following previous works, they strengthen the cross-attention from pixels inside the box to tokens associated with the box caption while attenuating the cross-attention from pixels outside the box. To achieve this, we define a simple energy function:

$$E\left(A^{(i)}, i, v\right)=-\operatorname{Topk}_u\left(A_{uv} \cdot \vec{b}^{(i)}\right)+\omega \operatorname{Topk}_u\left(A_{uv} \cdot\left(1-\vec{b}^{(i)}\right)\right) \tag{46}$$

where $\cdot$ is element-wise multiplication, $\vec{b}^{(i)}$ is a rectangular binary mask of the box i with the region in the box set to $1, \operatorname{Top}_u$ takes the average of top-k values across the spatial dimension u , and $\omega=4.0$. The energy function is minimized by updating the latent before each denoising step:

$$\begin{aligned}
& \vec{z}_t^{(i)} \leftarrow \vec{z}_t^{(i)}-\eta \nabla_{\vec{z}_t^{(i)}} \sum_{v \in V_i} E\left(A^{(i)}, i, v\right) \\
& \vec{z}_t^{(i)} \leftarrow \operatorname{Denoise}\left(\vec{z}_t^{(i)}\right)
\end{aligned}$$

Where $\eta$ is the guidance strength; the set $V_i$ contains the token indices for the box caption in the prompt for box i (e.g., while generating the masked latents for a box i with caption "a gray cat", $V_i$ indicates the indices of tokens that correspond to the box caption in the per-box denoising text prompt "[background prompt] with a gray cat"). Denoise $(\cdot)$ denotes one denoising step in the latent diffusion framework.

**insert figure here**

After generation, they obtain the cross-attention map that corresponds to the box caption, which serves as a saliency mask for the object.They optionally use SAM to refine the quality of the mask. This can be done by querying either with the pixel location that has the highest saliency or with the layout box. The functionality of SAM can also be replaced by a simple thresholding. With the refined mask for exactly one foreground instance, denoted as $\vec{m}^{(i)}$, they perform element-wise multiplication between the mask and the latent at each denoising step to create a sequence of masked instance latents: $\left\{\hat{z}_t^{(i)}\right\}_{t=0}^T$ :

$$\hat{z}_t^{(i)}=\vec{z}_t^{(i)} \otimes \vec{m}^{(i)} \tag{47}$$

Masked latents as priors for instance-level control. The masked instance latents $\left\{\hat{z}_t^{(i)}\right\}_{t=0}^T$ are then leveraged to provide instance-level hints to the diffusion model for the overall image generation. During each denoising time step in the early denoising process, we place each masked foreground latents $\hat{z}_t^{(i)}$ onto the composed latents $\vec{z}_t^{\text{comp}}$ :

$$\vec{z}_t^{\text{comp}} \leftarrow \operatorname{LatentCompose}\left(\vec{z}_t^{\text{comp}}, \hat{z}_t^{(i)}, \vec{m}^{(i)}\right) \forall \mathrm{i} \tag{48}$$

Where $\vec{z}_t^{\text{comp}}$ is initialized from $\vec{z}_T$ for foreground generation to ensure consistency, LatentCompose $\left(\vec{z}_t^{\text{comp}}, \hat{z}_t^{(i)}, \vec{m}^{(i)}\right)$ places the masked foreground latents $\hat{z}_t^{(i)}$ onto the corresponding locations on $\vec{z}_t^{\text{comp}}$. Since diffusion models typically determine object placement during the initial denoising steps and refine object details in later steps (Bar-Tal et al., 2023), the latents are composed only from timestep $T$ to $rT^3$, where $r \in[0,1]$ balances instance control and overall image coherence. By focusing on intervening during the object placement phase, the method provides instance-level layout hints without forcing the resulting generation to match the per-box generation exactly in each masked region.

To make the guidance more robust, the cross-attention maps from the per-box generation are transferred to the corresponding regions in the composed generation by adapting the energy function:

$$E^{\text{comp}}\left(A^{(\text{comp})}, A^{(i)}, i, v\right) = E\left(A^{(\text{comp})}, i, v\right) + \lambda \sum_{u \in V_i'} \left|A_{uv}^{\text{comp}} - A_{uv}^{i}\right| \tag{49}$$

where $\lambda = 2.0$, and the energy value of each box $i$ is summed for optimization. $V'$ represents the indices of tokens corresponding to the box caption in the text prompt during the denoising process, similar to the definition of $V_i$ in Equation (3). In this manner, the controller conditions the diffusion model to generate one instance at each masked location, ensuring that the final generation is both natural and coherent in terms of foreground-background composition.

Finally, the latent $\vec{z}_0^{(\text{comp})}$ is decoded into pixel space $\vec{x}_0$ using the diffusion image decoder.

The authors note the method's effectiveness but raise questions about its ability to handle interactions between various attributes. They suggest that its dynamic control over object creation could inspire further innovation, such as introducing an energy function to accelerate object generation. This would be analogous to an artist sketching the primary, challenging components of a piece first before adding background details.

## 2.11 Key-Value Injection

KV Injection (Key-Value Injection) is a technique used in diffusion models where features from a reference image are injected into the self-attention layers of a target image. This technique modifies the target image's appearance while preserving its structural layout, effectively transferring characteristics like color, pattern, or texture from the reference image onto the target.
Let $F_{\text{target}}$ be the feature map of the target image and $F_{\text{ref}}$ be the feature map of the reference image. In KV injection, the attention calculation uses:

The query from the target image $Q_{\text{target}}$ which encodes the spatial structure of the target.
The key and value from the reference image $K_{\text{ref}}$ and $V_{\text{ref}}$ which influence the target's appearance based on the reference.

The modified attention calculation becomes:

$$\text{Attention } = (Q_{\text{target}}\,, K_{\text{ref}}\,, V_{\text{ref}}\,) = \operatorname{softmax}\left(\frac{Q_{\text{target}}\, K_{ref}^{T}}{\sqrt{d}}\right) V_{\text{ref}} \tag{50}$$

The first paper that introduced this method was "MasaCtrl: Tuning-Free Mutual Self-Attention Control for Consistent Image Synthesis and Editing" [7]. Traditional text-to-image (T2I) generation models often fail to create consistent images when altering poses or views of the same object. In editing, these models can lose the original textures or identities of objects. MasaCtrl aims to solve this by transforming self-attention into a new type of mutual self-attention using source images as references to maintain visual consistency.

The authors utilize DDIM inversion to perform their edit. Let $\{P, P'\}$ be the original prompt and the modified prompt describing $\{I_S, I_t\}$ respectively. Note that there is no target image $I_t$ in the direct sense before the process. Instead, we would carry out the Masa-CTR pipeline to obtain our target image $I_t$ in accordance with the prompt $P'$

Given a source image $I_s$, we first encode it into a latent representation $\vec{z}_s$. To enable editing, the source latent $\vec{z}_s$ undergoes DDIM inversion conditioned on $P$. The result is $\vec{z}_S^T-$ a noise-infused latent representation that retains the "identity" of the source image but allows for flexible transformation in subsequent steps. We then take a latent $\vec{z}_t$ and noise it for T steps to obtain $\vec{z}_t^T$.

Finally, we run the forward pass to denoise $\vec{z}_t^T$ to $\vec{z}_t^0$ using the prompt $P'$. During this denoising process, instead of using keys $K_t$ and values $V_t$ from the target latent $\vec{z}_t^i$, the keys $K_s$ and values $V_S$ of the source latent $\vec{z}_s^i$ are utilized. This follows the key-value injection given by equation 50.

In practice, applying mutual self-attention across all layers and denoising steps would lead to the target image replicating the source image too closely. To control this:

- Mutual self-attention is applied selectively, usually in the decoder layers of the U-Net (where high-resolution details are formed).
- It is activated only after a certain denoising step S allowing the target image's layout to be initially guided by the target prompt.

Mathematically, the mutual self-attention mechanism is selectively applied as follows:

$$\text{EDIT } := \begin{cases} \text{MutualSelfAttention } (Q_t, K_S, V_S) \text{ if } t > S \text{ and } l > L \\ \text{SelfAttention } (Q, K, V) \text{ otherwise} \end{cases} \tag{51}$$

**Foreground-Background Confusion:** MasaCtrl struggles with foreground-background confusion because it fundamentally relies on the self-attention mechanism to transfer content (textures, colors, details) from the source image to the target image while allowing the target prompt to dictate structure (e.g., pose, layout). This approach has limitations when the foreground and background features are similar. Self-attention layers in diffusion models capture relationships between different regions of an image, but they don't inherently understand semantic boundaries (e.g., distinguishing a cat from a carpet when both are white). If the foreground and background have similar textures or colors, the model's attention maps might overlap or mix these regions, causing features to "bleed" from one area to another.

To overcome this, the authors utilize the cross-attention maps of diffusion model. When generating an image from text, cross-attention maps link regions of the image to specific tokens in the prompt. For example, in the prompt "a dog on grass," there would be cross-attention maps that highlight regions corresponding to "dog" and "grass.". These maps reflect which parts of the image are associated with each word in the prompt.

By averaging the cross-attention maps associated with these tokens across multiple attention heads and layers, we can generate a mask for the foreground (e.g., the dog) and a separate mask for the background (e.g., the grass). With separate masks for the foreground and background, mutual self-attention can selectively apply source content to the appropriate areas.

Let M be the foreground mask M for the source and target image and $1 - M$ be the complement of the foreground mask. Then the mutual self-attention outputs for the foreground and background regions can be computed separately:

Foreground Self-attention:

$$f_{\text{foreground }} = \text{Attention}\left(Q, K_S, V_s; M\right) = \text{sofmax}\left(\frac{QK_s^T}{\sqrt{d}}\right) V_S.M \tag{52}$$

This output focuses on the foreground area and applies only to regions where M = 1 (foreground):

$$f_{\text{background }}=\operatorname{Attention}\left(Q, K_S, V_s ; 1-M\right)=\operatorname{sofmax}\left(\frac{Q K_s^T}{\sqrt{d}}\right) V_s .(1-M) \tag{53}$$

The combined output assumes the following form:

$$f=f_{\text{background }}+f_{\text{foreground }}$$

This combined attention output ensures that the model retains clear distinctions between foreground and background areas with each area sourcing only the relevant content from the source image.

**Using Key-value Injection for Appearance Transfer:** The paper "Cross-Image Attention for Zero-Shot Appearance Transfer" [2] uses KV-injection but for different purposes. While MasaCtrl focuses on mutual self-attention control within a single image, the cross-image attention mechanism here is designed to handle two separate images: one for structure and one for appearance. The approach they utilize transfer appearance features from one image onto another while respecting the layout or pose of the target image.

Suppose we have a structure image of a giraffe with long legs and a distinct neck structure. We also have a appearance image with a zebra with stripped patterns. The queries of Istruct (giraffe's structure) interact with keys and values from Iapp (zebra's appearance). For instance, a query on the giraffe's neck might attend to the zebra's body, aligning their semantic roles (e.g., elongated neck/striped body). The attention map is adjusted to enhance specific correspondences (e.g., giraffe neck aligns sharply with zebra body).

The KV-injection assumes the form:

$$\text{Attention }_{\text{cross }}=\left(Q_{\text{struc }}, K_{\text{app }}, V_{\text{app }}\right)=\operatorname{softmax}\left(\frac{Q_{\text{struct }} K_{\text{app }}^T}{\sqrt{d}}\right) V_{\text{app }} \tag{54}$$

To sharpen the attention maps and ensure that cross-image attention maps are distinct and focused, they introduce a contrast enhancement step. This process increases the variance in attention weights, which helps the model attend more distinctly to relevant regions.

Classifier-free guidance is used to further align the appearance transfer with the desired characteristics of the target image. This technique, originally developed for conditional generation, is adapted here to control the extent to which appearance features from $I_{app}$ influence the final output. For appearance guidance, they modify the cross-image attention by blending it with unconditioned (structure-only) attention:

$$\text{Attention }_{\text{guided }}=(1+w) . \text{ Attention }_{\text{cross }}-w \cdot \text{ Attention }_{\text{uncond }} \tag{55}$$

The interesting thing that the authors utilize is AdaLN where the feature maps $F_{\text{struct }}$ are adjusted to match the appearance statistics of appearance features $F_{\text{app }}$. The transformation reads:

$$\operatorname{AdaLN}\left(F_{\text{struct }}, F_{\text{app }}\right)=\sigma\left(F_{\text{app }}\right)\left(\frac{F_{\text{struc }}-\mu\left(F_{\text{struct }}\right)}{\sigma\left(F_{\text{struct }}\right)}\right)+\mu\left(F_{\text{app }}\right) \tag{56}$$

**Eye-for-an-eye: Appearance Transfer with Semantic Correspondence in Diffusion Models**: The paper Eye-for-an-eye: Appearance Transfer with Semantic Correspondence in Diffusion Models [15] builds on the previous works. It introduces a systematic way to transfer the appearance from a reference image to a target image while ensuring that the transfer respects the semantic regions of the target image.

During each timestep of the diffusion process, feature maps are extracted from specific layers of the U-Net for both the target image and reference image. For each pixel $q$ in the target image feature map $F_{\text{target}}$ , the method finds a corresponding pixel $p$ in the reference image feature map $F_{\text{ref}}$ by maximizing their cosine similarity. This finds the pixel in $F_{ref}$ that best matches the appearance of q in $F_{\text{target}}$ , aligning textures based on content rather that proximity. Mathematically,

$$p = \arg \max_{p \in [0,h] x [0,w]} \operatorname{sim} \left( F_{\text{target}} (q), F_{\text{ref}} (p) \right) \tag{57}$$

This similarity-based matching creates a semantic alignment between features in the two images, enabling textures or colors to map onto corresponding areas (e.g., a zebra's stripes onto a giraffe's body). Once the correspondences are established, the reference features are rearranged to match the spatial arrangement of the target image, resulting in $\widetilde{F}_{ref}$. This realignment ensures that the transferred features map semantically to the correct regions in the target image.

To integrate the rearranged features into the target image, they use a mask that selectively transfers appearance only to relevant regions of the target. This mask helps separate the foreground and background to prevent background features from contaminating the object features (e.g., applying sky texture onto an animal's body).

$$F_{\text{masked}}^{\text{target}} = F_{\text{target}} \left[ M_{\text{target}} \right], F_{\text{masked}}^{\text{ref}} = F_{\text{ref}} \left[ M_{\text{ref}} \right] \tag{58}$$

Here, only the masked regions are considered for matching, ensuring that only semantically meaningful features are aligned. After obtaining semantically aligned features, the method injects them back into the target image feature map:

$$F_{\text{out}}' = F_{\text{ref}}^{\sim} \odot M_{\text{target}} + F_{\text{out}} \odot \left( 1 - M_{\text{target}} \right) \tag{59}$$

Where $F_{ref}^{\sim}$ represents the rearranged reference features based on semantic matching. The modified self-attention then uses:

$$\text{Self-attn} \left( F_{\text{out}}' \right) = \operatorname{softmax} \left( \frac{Q_{\text{out}}' \left( K_{\text{out}}' \right)^{T}}{\sqrt{d_k}} \right) V_{\text{out}}' \tag{60}$$

Where $Q_{\text{out}}'$ , $K_{\text{out}}'$ and $V_{\text{out}}'$ are derived from $F_{\text{out}}'$ . Finally, Adaptive Instance Normalization (AdaIN) is applied to balance the brightness and color contrast between the reference and target features:

$$\operatorname{AdaLN}(F) = \sigma_{\text{ref}} \frac{\left( F - \mu_{\text{target}} \right)}{\sigma_{\text{target}}} + \mu_{\text{ref}} \tag{61}$$

This standardizes the appearance features of the target with the mean and variance from the reference, smoothing out color and brightness discrepancies.

# 3 Personalization:

## 3.1 Future Works and Ideas

1. Read ProSpect: Prompt Spectrum for Attribute-Aware Personalization of Diffusion Models
2. Read Dreambooth 3. Read the CMU work where they aim to perform customization in one-shot. 4. Read "Multi-Concept Customization of Text-to-Image Diffusion 5. Isn't personalization in some sense capturing the manifold of the embedding? can't we introduce some sort of noise methodology to ensure that the embeddings learned are robust? 6. Make an extensive section on editability vs distortion 7. With **"Break-A-Scene"**, can't we combine unsupervised segmentation methods like **DiffAttend** and correspondence method to improve personalization? For example, suppose that I have a collection of images $I$ consisting of a person. If I am able to perform unsupervised segmentation and correspondense to focus on "eye", I can train a specific embedding to capture "eye". 8. Incorporate Reversion II also. 9. Read "A Neural Space-Time Representation for Text-to-Image Personalization"

## 3.2 GAN-approaches

Personalization as a concept predates the emergence of diffusion models and finds its origins in advancements made with GANs. Early approaches in context of GANs focused on adapting pretrained models to generate content specific subjects or styles. Among these, Few-Shot GAN Adaptation [23] introduced a framework for fine-tuning GANs on small datasets (e.g., 10-20 images) without overfitting by using Elastic Weight Consolidation (EWC). This technique identified the most critical weights in the pre-trained GAN and penalized large updates to them. This ensured the model generates subject-specific images while retaining its ability to generate diverse outputs. Similarly, Instance-Conditioned GANs extended GANs by conditioning the generation process on instance-level embeddings. This enabled the model to synthesize variations of specific objects or subjects [8]. These methods were effective for generalization across similar instances but often lacked the fine-grained detail required for unique self-defined subjects.

The introduction of StyleGANs [21] significantly advanced the personalization paradigm. It gave rise to latent space manipulation [1] and the subsequent usage of pivotal tuning [27] to further optimize the former methodology and address the weaknesses arising from it. These approaches exploited the disentangled latent space of StyleGAN to allow detailed subject-specific edits and stylistic adjustments. Since both of these methodologies share similarities with the subsequent approaches used to perform personalization in Diffusion Models, a closer examination of both of the approaches will be help.

To begin with, let us give a high overview of StyleGANs. In traditional GANs, the latent vector $\vec{z}$ is sampled from a fixed distribution and directly fed into the generator's input layer. The generator then transforms $\vec{z}$ into an image using a series of fully connected and convolutional layers. This direct mapping from $\vec{z}$ to image $\mathcal{J}$ lead to entangled and poorly separated features in the latent space. StyleGAN introduces an intermediate latent space $\vec{w}$ which is a transformed version of $\vec{z}$. Instead of directly using $\vec{z}$, it passes $\vec{z}$ through a mapping network f to produce $\vec{w}$, which controls the style at different layers of the generator. Therefore, instead of the direct map:

$$G : \vec{z} \to \mathcal{J}$$

We now have an intermediate map f , usually in the form of MLP:

$$f : \vec{Z} \rightarrow \vec{w}$$
$$G : \vec{w} \rightarrow \mathcal{J}$$

The mapping network helps disentangle features and provides better control over different aspects of the image. For example, $\vec{w}$ might control high-level attributes like pose at some layers and fine details like color at others. This leads to a more interpretable and manipulatable latent space.

The structure of StyleGANs is shown below:

**INSERT FIGURE HERE**

StyleGAN introduces Adaptive Instance Normalization (AdaIN) to gain fine-grained control over the image at multiple levels of abstraction. The style vector $\vec{w}$, after being transformed by the mapping network $f(\vec{z})$, is applied to each layer of the generator through AdaIN, which adjusts the mean and variance of the feature maps:

$$\text{AdaLN}(\vec{x}, \vec{w}) = \sigma(\vec{w}) \frac{\vec{x} - \mu(\vec{x})}{\sigma(\vec{x})} + \mu(\vec{w})$$

Where $\vec{x}$ is the feature map and $\vec{w}$ controls the style at each resolution level.
In StyleGAN, random noise is added at each layer of the generator independently from the latent vector $\vec{w}$. This noise is injected into the feature maps to introduce stochastic variations in the generated images, particularly affecting fine details like hair strands or skin texture:

$$\vec{x}' = \vec{x} + N$$

Where N is the noise map that adds variation to the feature map $\vec{x}$. To promote disentanglement, StyleGAN uses style mixing regularization during training. It randomly selects two latent vectors $\vec{w}_1$ and $\vec{w}_2$ and applies them at different layers of the generator. For example, $\vec{w}_1$ might control the style at the lower-resolution layers (coarse features), and $\vec{w}_2$ might control the higherresolution layers (fine details):

$$G\left(\text{AdaLN}\left(\vec{x}_1, \vec{w}_1\right), \text{AdaLN}\left(\vec{x}_2, \vec{w}_2\right)\right) \rightarrow \text{ Mixed Styles}$$

Given the structure of Style GANs, we can understand Latent Space Manipulation as introduced by R Abdal et al in 2019. In StyleGAN, there are multiple latent spaces where an input vector can be embedded and these latent spaces are essential for controlling the style and content of the generated images. Z is the initial latent space, where an input vector $\vec{z}$ is sampled from a simple distribution, typically a Gaussian distribution $\mathcal{N}(0, \mathbb{I})$. This space is 512-dimensional. However, it is extremely entangled.

As we saw, W is the intermediate latent space in StyleGAN. The vector $\vec{w} \in$ W is obtained by passing $\vec{z}$ through a learned, fully connected neural network called the mapping network f as we saw above. However, while this space is better suited for controlling style at a high level, it doesn't give fine-grained control at individual layers of the generator. Thus, we see that while Z is too entangled, W space comes with its own limitations: it applies the same latent vector $\mathrm{w}^{\vec{Z}}$ across all layers of the generator. This means that one latent vector controls every layer, limiting the flexibility to adjust styles independently at different layers.
W + is an extended latent space introduced to overcome the limitations of embedding directly into

W. Instead of using a single latent vector $\vec{w}$ for the entire generator, W+ concatenates 18 different 512-dimensional latent vectors (one for each layer of the StyleGAN generator that receives input via AdaIN):

$$W^{+}=\left\{w_1, w_2, \ldots w_{18}\right\}, w_i \in \mathbb{R}^{512}$$

Where the total number of layers in StyleGANs is 18. Each layer now receives a separate latent vector $\vec{w}_i$ rather than all layers being controlled by the same $\vec{w}$. The latent space, $\boldsymbol{W}+$ allows for more layer-specific control over the generated image, meaning you can independently control coarse features (e.g., pose) in earlier layers and finer features (e.g., texture, color) in later layers.

Given the $W^{+}$space, the authors solve for a latent code $\vec{w}^*$ such that the generate image $G\left(\vec{w}^*\right)$ closely resemble the input image $\mathcal{J}$. This is achieved by minimizing a reconstruction loss:

$$\vec{w}^*=\arg \min_{w \in W} \mathcal{L}_{\text{total}}\left(G(\vec{w}), \mathcal{J}\right)$$

Where $\mathcal{L}_{\text{total}}$ includes pixel-wise loss and perceptual loss. Note that the process does not involve the fine-tuning of the Generator $G$ which remains frozen.

While latent-space manipulation achieves editability due to the disentangled nature of $W^{+}$, this limits reconstruction fidelity. The authors of the paper [30] encapsulate this tension between reconstructing a real image and retaining the ability to perform semantic edits in what they call the ”Distortion vs Editability” trade-off. In W space, the generator has low reconstruction fidelity but high editability. On the other hand for $W^{+}$, we have high fidelity but low editability. The reason for that is that W's generator's strong priors (a single collection of weight is optimized throughout) ensure edits are meaningful and disentangled, but reconstructions suffer because W lacks fine-grained details. With $W^{+}$, it is the opposite: the generator can better reconstruct real images, but edits become less predictable due to the weights being disentangled.

In Pivotal Tuning [27], the aim is to resolve this ”Distortion vs Editability” trade-off to balance both fidelity and semantic editability. To carry this out, the authors fine-tune the generator $G_\theta$ around the pivot latent code $\vec{w}_p$ :

$$\theta^*=\arg \min_\theta \mathcal{L}_{PTI}=\mathcal{L}_{\text{Recon}}\left(G_\theta\left(w_p\right), \mathcal{J}\right)+\lambda_{\text{reg}} \mathcal{L}_{reg}(\theta)$$

By fine-tuning only locally around $w_p$, pivotal tuning preserves the disentangled structure of $W$ and $W^{+}$, ensuring semantic directions remain well-defined.

Another notable work in context of personalization is [26]. The methodology enables text-driven personalization of StyleGAN-generated images by aligning the StyleGAN latent space with CLIP's multimodal text-image embedding space. Through optimization or a trained latent mapper, it manipulates latent vectors to match semantic descriptions in text while preserving key attributes like identity or structure. This approach allows users to perform intuitive, fine-grained edits to images based solely on textual guidance.

### 3.3 DreamBooth

Unlike textual inversion which we would study later, DreamBooth [28] fine-tunes **the entire text-to-image diffusion model** embedding the subject directly into the output domain rather than limiting it to the latent embedding space. The pretrained model parameters $\theta$ are fine-tuned to bind a unique identifier $[V]$ with subject-specific features. While the embedding $c$ remains consistent with Textual Inversion, it is now utilized to fine-tune the model itself.

We begin with 3-5 images of the subject captured in varying contexts. These images are labeled with descriptive text prompts such as *”a [V] dog”*, where $[V]$ is a unique identifier that represents the subject. To ensure uniqueness, $[V]$ is chosen as a token rarely present in the model’s vocabulary, minimizing interference with existing knowledge.

Let the collection of images $\{\mathcal{J}_1, \mathcal{J}_2, \dots \mathcal{J}_5\}$ contain the concept that we would like to personalize. The noise latent $\{\vec{z}_1, \vec{z}_2, \dots \vec{z}_5\}$ are obtained by passing each image to the encoder of VQ-VAE and then performing the forward diffusion process. We then introduce a learnable pseudo-word $[V]$. The model U-NeT must denoise the latents back into a meaningful image conditioned on the text input that contains the pseudo-word $[V]$.

The text-to-image diffusion model (e.g., Stable Diffusion or Imagen) is fine-tuned on the subject’s images and their corresponding descriptive prompts. The fine-tuning process uses the diffusion model loss:

$$\mathcal{L}_{\text{noise}} = \mathbb{E}_{x,c,\epsilon,t}\left[w_t \left\|\hat{\epsilon}_\theta(\alpha_t x + \sigma_t \epsilon, c) - \epsilon\right\|_2^2\right] \tag{62}$$

where the loss is backpropogated to directly optimize the parameters $\theta$ of the model. This step trains the model to associate $[V]$ with the specific visual characteristics of the subject while preserving its knowledge of what a ”dog” generally looks like.

Fine-tuning on a small set of images risks overfitting, where the model might either memorize the exact training images or lose its ability to generate diverse outputs for the subject class (e.g., other ”dogs”). To mitigate this, a class-specific prior preservation loss is introduced. This loss supervises the model using both the fine-tuning dataset (specific to the subject) and synthetic samples generated by the pretrained model for the general subject class. The prior preservation loss is:

$$\mathcal{L}_{\text{noise}} = \mathbb{E}_{x,c,\epsilon,\epsilon',t}\left[w_t \left\|\epsilon - \hat{\epsilon}_\theta(\alpha_t x + \sigma_t \epsilon, c)\right\|_2^2 + \lambda w'_t \left\|\epsilon' - \hat{\epsilon}_\theta(\alpha'_t x_{\text{pr}} + \sigma'_t \epsilon', c_{\text{pr}})\right\|_2^2\right]. \tag{63}$$

where the first term ensures fidelity to the specific subject while the second term encourages diversity by using the pretrained model’s outputs as pseudo-labels. For example, suppose we are personalizing a particular dog. Then, $c_r$ would be ”dogs”. The model predicts the noise $\epsilon'$ for the generic dog and learns to reconstruct it. This prevents the model from overfitting to the personalized concept and losing the broader understanding of ’dogs’.

After fine-tuning, the model can synthesize the subject in new scenarios using simple text prompts such as *”A [V] dog in the jungle”* or *”A statue of a [V] dog in the style of Michelangelo”*. The model leverages:

- The unique visual features tied to $[V]$ (e.g., the white patch on the dog’s forehead).
- Its extensive prior knowledge about jungles, statues, and artistic styles.

This enables the creation of photorealistic and artistic renditions of the subject in diverse contexts.

## 3.4 Textual Inversion for Diffusion Models

### 3.4.1 An Image is worth one word

[13] introduces textual Inversion for Diffusion models. This allows a user to represent a unique concept (e.g., a favorite toy, a personal object, etc.) by finding new pseudo-words in the embedding space of a pre-trained text-to-image model. These pseudo-words are optimized to guide the text-to-image model into generating faithful representations of the unique concept.

In the stable diffusion II pipeline, the text prompt t is tokenized into individual tokens $\{\vec{w}_1, \dots \vec{w}_n\}$ and each token is embedded using CLIP text encoder. Each token $\vec{w}_i$ maps to a corresponding embedding vector $\vec{e}_i \in \mathbb{R}^d$ in the latent space, where d is the dimensionality of the embedding. The overall text embedding for the prompt is often denoted as:

$$E(t) = [\vec{e}_1, \dots \vec{e}_n] \in \mathbb{R}^{nxd} \tag{64}$$

This embedding sequence conditions the image generation process, influencing attention maps in the U-Net or other architectures.

In Textual Inversion, you introduce new tokens that don't exist in the original vocabulary of the model. These tokens $\vec{w}_{\text{new}}$ represent a new concept or object. Instead of randomly initializing $\vec{e}_{\text{new}}$ for these new tokens, Textual Inversion learns an embedding vector for $\vec{w}_{\text{new}}$ by optimizing it based on a few images of the concept you want to learn.

Given a collection of images $\{\mathcal{J}_1, \mathcal{J}_2, \dots \mathcal{J}_5\}$ containing the concept that we would like to invert, the noise latent $\{\vec{z}_1, \vec{z}_2, \dots \vec{z}_5\}$ are obtained by passing each image to the encoder of VQ-VAE and then performing the forward diffusion process. We then introduce a learnable pseudo-word $\vec{d}$ in the template prompts **D**. The model U-NeT must denoise the latents back into a meaningful image conditioned on the text input that contains the pseudo-word $\vec{d}$ :

$$\mathcal{L}(\vec{d}) = \mathbb{E}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{z}_t, \vec{d}\right)\right\|_2^2\right]$$

This is a reconstruction loss where given the template prompt **D** and a latent $\vec{z}_i$, the model must reconstruct $\mathcal{J}_i$. Initially, the vector is a poor representation of the target concept so the model won't be able to correctly predict the noise associated with the images of that concept, resulting in a higher loss. The optimization adjusts $\vec{d}$ so that the model becomes better at predicting the noise in the latent space when generating images that match the concept represented by $\vec{d}$

The template prompts **D** for a *single subject* assume the following form, to state a few of them:

- 'a photo of a } ', 'a rendering of a } ', 'a cropped photo of the } ', 'the photo of a } ',
- 'a photo of a clean } ', 'a photo of a dirty } ', 'a dark photo of the } ', 'a photo of my } ',
- 'a photo of the cool } ', 'a close-up photo of a } ', 'a bright photo of the } '

, Where "}" is the place-holder. On the other hand, the template prompts **D** for a *dual subject* assume the following form:

- 'a photo of a } with } ', 'a rendering of a } with } ', 'a cropped photo of the } with }′,
- 'the photo of a } with } ', 'a photo of a clean } with } ', 'a photo of a dirty } with } ',
- 'a dark photo of the } with } ', 'a photo of my } with } ', 'a photo of the cool } with }′,

This optimization process finds an embedding $\vec{d}$ that represents the user-provided concept, and it can then be reused in any text prompt for generating new images. The authors experiments were conducted using 2× V100 GPUs with a batch size of 4 The following method requires 2-3 hours of training on a GPU. Therefore, the entire framework can *easily be carried out by an independent researcher with low-compute resources.* Multiple code implementations are available. For example, see the following Hugging Face tutorial

When it came to experiments, the authors considered four different variations for performing textual inversion:

- **Approach 1:** Instead of using a single embedding vector $\vec{d^*}$ to represent the concept, the authors use multiple vectors to represent the same concept. The idea is to describe the concept through multiple learned pseudo-words, such as using two vectors for a 2-word setup or three vectors for a 3 word setup. The rationale is that the single-vector setup might act as a bottleneck because a single vector could struggle to capture all the fine details of a complex concept. By using multiple vectors, the model can potentially capture a richer representation of the concept, leading to more accurate image reconstructions.

- **Approach 2:** The authors used Progressive extensions. Instead of starting with multiple vectors from the beginning, the authors start with one vector and add more vectors gradually during training. Specifically, they start with a single vector, then add a second vector after 2000 steps, and a third vector after 4000 steps. The idea is to allow the model to first focus on capturing the core details of the concept with one vector. Once those details are learned, additional vectors are introduced to capture finer details of the concept. This staged approach could help the model avoid being overwhelmed by too many vectors from the start and focus on the essential aspects of the concept first.

- **Approach 3:** The authors apply a regularization term that encourages the learned embedding vector $\vec{d^*}$ to stay close to the embeddings of existing words in the pre-trained model's vocabulary. In practice, this is done by minimizing the L2 distance between the learned embedding and the embedding of a coarse descriptor of the object (e.g., if the object is a sculpture, the coarse descriptor might be the word ”sculpture”). This method is inspired by observations in GAN inversion that latent codes which stay closer to the distribution of real-world embeddings (those seen during training) have increased editability. By keeping the learned embedding close to an existing word embedding, the model might become more flexible in editing the concept while preserving the ability to generate realistic images.

- **Approach 4:** In this method, the authors introduce unique tokens for each image in the training set, rather than using a single word embedding for the entire set of images. In addition to a shared placeholder $S_i$ for the concept, they introduce unique placeholders $S_i$ for each image. For example, the concept $S^*$ (the object) is shared across images, while each image i is associated with a unique embedding $S_i$ capturing per-image details like background variations.

### 3.4.2 Extended Textual Inversion

In the work [13], a single embedding **p** is embedded into the U-net. This embedding interacts with the image features $\vec{f_t}$ using cross-attention in the U-NeT across all the layers $l = 1, ...n$. In the work [33] the authors take inspiration from previous works in context of GANs, namely [1] which introduces

W+ space to expand the traditional W space. Instead of sending a single embedding **p** into the U-NeT, they send each of the embedding $\{p_1, p_2, \dots p_n\}$ to a particular layer where each layer $p_i \in P$ is specific to the i-th layer of the U-Net. Thus for the prompt p =a red cat and a blue dog, we might have that $p_2 =$ "red" is sent to the first layer, $p_3 =$ "cat" is sent to the second layer and so on...

Usually, a single embedding vector $p^*$ is optimized in personalization. However with the above paradigm, instead of using a single embedding vector $p^*$ for all layers, we can learn multiple embeddings $\{p_1, p_2, \dots p_n\}$ where each layer $p_i \in P$ is specific to the i -th layer of the U-Net. Each layer learns a different embedding that focuses on certain aspects of the image-coarse layers handle structure, while fine layers handle appearance.

To introduce their methodology, the authors carry out a simple experiment on the publicly available Stable Diffusion model. They partitioned the cross-attention layers of the denoising U-net into two subsets: coarse layers with low spatial resolution and fine layers with high spatial resolution. They then used two conditioning prompts: "red cube" and "green lizard", and inject one prompt into one subset of cross-attention layers, while injecting the second prompt into the other subset. Notably, at the first run the model generates a red lizard, by taking the subject from the coarse layers' text conditioning, and appearance from the fine layers' conditioning. Similarly, in the second run it generates the green cube, once again taking the appearance from the fine layers and the subject from the coarse layers. This experiment suggests that the conditioning mechanism at different resolutions processes prompts differently, with different attributes exerting greater influence at different levels

Given a set of images $\mathcal{J} = \{I_1, \dots I_k\}$ of a specific subject, the goal of the Textual Inversion (TI) operation is to find a representation of the object in the conditioning space P . In **Extended Textual Inversion (XTI)**, we add n new textual tokens $t_1, \dots, t_n$ to the tokenizer model, associated with n new token embeddings lookup-table elements $e_1, \dots, e_n$. Then, we optimize the token embeddings with the objective to predict the noise of a noisy images from $\mathcal{J}$, while the token embeddings are injected to the network.

Assuming that the denoising U-net is parameterized by a set of parameters denoted by $\theta$, and operates within the extended conditioning space as previously described, the reconstruction objective for the embeddings $e_1, \dots, e_n$. that correspond to the tokens $t_1, \dots, t_n$ as follows:

$$\mathcal{L}_{XTI}\left(v^*\right) = \mathbb{E}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{z}_t, p_1, p_2, \dots p_n\right)\right\|_2^2\right] \tag{65}$$

Where $\vec{z}_t$ is the image $\mathcal{J}$ noised with the additive noise $\varepsilon$ according to the noise level t . This optimization is applied independently to each cross-attention layer.

To understand the framework, suppose we want to personalize a model to generate images of a customized teapot. For example, **a blue teapot with unique floral patterns** while retaining flexibility to manipulate its **appearance** or **shape**. With P$^+$, the conditioning embeddings are *layer-specific*:

$$P^+ = \{p_1, p_2, \dots, p_n\}$$

where $p_i \in \mathbb{R}^d$ (dimensionality $d$) corresponds to the conditioning embedding for the $i$-th cross-attention layer of the U-Net. The coarse layers $p_1, \dots, p_k$ control shape/structure while Fine layers $p_{k+1}, \dots, p_n$ control *appearance/details.*

In XTI process, we add new textual tokens $t_1, t_2, \dots, t_n$ corresponding to each layer rather a single textual token and we optimize embeddings $e_1, e_2, \ldots, e_n$ to reconstruct the image $I$. We now have a set of layer-specific embeddings $\{e_1, e_2, \ldots, e_n\}$ that encode our blue floral teapot into the P$^+$ space. We can now use these embeddings $\{e_1, e_2, \ldots, e_n\}$ to generate images with layer-wise control. Suppose we want to keep the **shape** of the teapot but change its **appearance** to *golden*. For that, we would inject the shape embeddings $e_1, \ldots, e_k$ (coarse layers) from our personalized teapot and replace the appearance embeddings $e_{k+1}, \ldots, e_n$ (fine layers) with those from a prompt like "golden teapot. The generated image retains the custom shape of the blue floral teapot but now has golden color. On the other hand, we we want to the **appearance** of the floral patterns but modify the **shape** to a cube then we would inject shape embeddings $c_1, \ldots, c_k$ (from "cube") into the coarse layers and keep the appearance embeddings $e_{k+1}, \ldots, e_n$ (from the personalized teapot).

On a broader level, TXI improves the **representation of embeddings** by disentangling the independent variations in the underlying concept c that we would like to embed.

**Using TXI for Style Mixing**

An interesting application of TXI framework is style-mixing. Given two concepts $A$ (e.g., a skull mug) and $B$ (e.g., a golden cat statue), we can generate a new image that combines the **geometry** of concept $A$ and the **appearance** of concept $B$.

We assume that XTI has already been applied to both concepts, resulting in two sets of optimized embeddings:

$$P_A^+ = \{e_{A1}, e_{A2}, \ldots, e_{An}\}, \quad \text{Layer-wise embeddings for concept } A.$$
$$P_B^+ = \{e_{B1}, e_{B2}, \ldots, e_{Bn}\}, \quad \text{Layer-wise embeddings for concept } B.$$

The idea is to inject the shape embeddings and the appearance embeddings $e_{B(k+1)}, e_{B(k+2)}, \ldots, e_{Bn}$ from fine layers of $B$). Up until k layer, we would inject shape embeddings and after that we would inject concept embeddings. This would lead to a mixed embedding $P^+_{\text{mix}}$. When we would condition our T2I diffusion model on this embedding, the resulting image will exhibit shape from concept A and appearance from concept B.

The style mixing process can be generalized to blend multiple concepts. For example, by blending embeddings at different ranges $k$ and $K$, we can control how much of the shape or appearance is borrowed from each concept.

### 3.4.3 Break-A-Scene

The methods discussed so far aim to introduce a user-provided concept to the model which allows its synthesis in diverse contexts. However, they primarily focus on the case of learning a single concept from multiple images with variations in backgrounds and poses. In [3], the authors introduce effective textual inversion given a **single image** of a scene that may **contain several concepts**. To do so, the authors augment the input image with masks provided by the user or by a pre-trained segmentation model. They then perform two-phase customization processes that optimizes a set of dedicated textual embeddings and the model weights:

1. Token Optimization (Phase 1)

2. Weights Optimization (Phase 2)

Note that the above does not strictly **obey the textual inversion paradigm**. Instead, both the model parameters and token is being optimized so it is a mix of both methodologies - something that would be characteristic now of many of the works that we will discuss.

Suppose we have an input image $I$. The methodology introduces a set of $N$ masks $\{M_i\}_{i=1}^N$ indicating regions of interest in the image. The goal is to extract $N$ textual embeddings $\{v_i\}_{i=1}^N$, referred to as handles, such that each $v_i$ represents the concept in mask $M_i$ and that these handles can be combined in various textual prompts to generate new images or novel combinations of the concepts.

During **token optimization**, the model weights are frozen and the textual embeddings $v_i$ are optimized to reconstruct the input image $I$ using a masked diffusion loss:

$$L_{\text{rec}} = \mathbb{E}_{z,t,\epsilon\sim\mathcal{N}(0,1)}\left[\left\|\epsilon \odot M_s - \epsilon_\theta(z_t, t, p_s) \odot M_s\right\|_2^2\right], \tag{66}$$

where $M_s = \bigcup_{i\in s} M_i$ is the combined mask for the selected concepts. For example, for an image representing a dog and a cat, we would generate masks for dog and cat respectively. During training, we would denoise the entire image, but calculate the loss only for how well the noise is estimated at masked region.

During **Weights Optimization**, both the handles $\{v_i\}$ and model weights $\theta$ are fine-tuned with a smaller learning rate to improve fidelity without overfitting.

To ensure that the model can synthesize combinations of concepts, a union sampling strategy is employed. At each training step, a random subset $s \subseteq \{1,\ldots,N\}$ of concepts is selected. A prompt is constructed, e.g., `‘‘a photo of [`$v_1$`] and [`$v_2$`],’’` and the corresponding combined mask $M_s$ is used for the diffusion loss. This ensures that the handles are jointly trained to generate individual concepts and their combinations.

Another novel feature of the work is the introduction of **cross-attention loss** for disentanglement:

$$L_{\text{attn}} = \mathbb{E}_{z,t}\left[\left\|\text{CA}_\theta(v_i, z_t) - M_i\right\|_2^2\right] \tag{67}$$

where $\text{CA}_\theta(v_i, z_t)$ is Cross-attention map between handle $v_i$ and noisy latent $z_t$. The loss measures how well the cross-attention maps align with their corresponding masks, ensuring that the model is focusing on the specific object of interest. The total loss becomes:

$$L_{\text{total}} = L_{\text{rec}} + \lambda_{\text{attn}} L_{\text{attn}} \tag{68}$$

where $\lambda_{\text{attn}}$ is a weighting factor.

### 3.4.4 Reversion

While the methods discussed so far aim to capture objects, the work [20] extends textual inversion to encoder higher-order concepts such as relationships between objects. The work design a novel relation-steering contrastive learning scheme to steer the relation prompt towards a relation-dense region in the text embedding space. They use a set of basis prepositions as positive samples to pull the embedding into sparsely activated regions while treating words from other parts of speech (e.g., nouns, adjectives) in text descriptions as negative samples. This approach helps disentangle semantics related to object appearances. To further emphasize object interactions, the authors propose a relation-focal importance sampling strategy, which constrains the optimization process to prioritize high-level interactions over low-level details.

Appearance inversion focuses on inverting low-level features of a specific entity, thus the commonly used pixel level reconstruction loss is sufficient to learn a prompt that captures the shared information in exemplar images. In contrast, relation is a high-level visual concept. A pixelwise loss alone cannot accurately extract the target relation. Some linguistic priors need to be introduced to represent relations.

The authors present the **preposition prior**, a language-based prior that steers the relation prompt towards a relation-dense region in the text embedding space. This prior is motivated by a well-acknowledged premise and two interesting observations on natural language. The first premise of the authors is that **Prepositions describe relations**. In natural language, prepositions are words that express the relation between elements in a sentence. To back this premise, they note the **POS clustering** in the CLIP space. Embeddings are generally clustered according to their Part-of-Speech (POS) labels in CLIP. This observation motivates the authors to steer the relation prompt $\langle R\rangle$ towards the preposition subspace (i.e., the red region in Figure below)

**INSERT FIGURE**

As illustrated in the figure below, the feature similarity between a real-world relationship and prepositional words exhibits a sparse distribution. The prepositions that are activated tend to align closely with the semantic meaning of the given relationship. For instance, in the case of the relationship ”swinging,” prepositions such as ”underneath,” ”down,” ”beneath,” and ”aboard” are sparsely activated, collectively capturing the essence of the ”swinging” interaction. This **second observation** highlights that only a specific subset of prepositions should be activated during the optimization process, which forms the basis of our noise-resistant design.

Similar to pseudo-word prompt, a relation prompt $\langle R\rangle$ is optimized using the reconstruction loss:

$$\langle R\rangle = \operatorname{argmin}_{\langle r\rangle} \mathbb{E}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{x}_t, \tau_\theta(\vec{c})\right)\right\|^2\right] \tag{69}$$

As the above discussion highlights, the above loss mainly focuses on pixel-level reconstruction rather than visual relation. In order to learn the more general concept of ”relations” between objects, the authors utilize the embedding space of clip. They define propositions as positive samples and other POS’ words (Nouns, adjectives) as negative samples to construct the following loss:

$$L_{\text{pre}} \ = -\log \frac{\exp\left(\frac{R^T \cdot P_i}{r}\right)}{\exp\left(\frac{R^T \cdot P_i}{r}\right) + \sum_{k=1}^{K} \exp\left(\frac{R^T \cdot N_i}{r}\right)} \tag{70}$$

Furthermore, only a small set of propositions should be considered as true positives. Thus, 70 needs to be defied as following:

$$L_{\text{steer}} \ = -\log \frac{\sum_{l=1}^{L} \exp\left(\frac{R^T \cdot P_i^l}{r}\right)}{\exp\left(\frac{R^T \cdot P_i}{r}\right) + \sum_{k=1}^{K} \exp\left(\frac{R^T \cdot N_i}{r}\right)} \tag{71}$$

Where $P_i = \left\{P_i^1, \ldots P_i^L\right\}$ refers to positive samples randomly drawn from a set of basis prepositions and $N_i = \left\{N_i^1, \ldots N_i^M\right\}$ refers to the improved negative samples. The above is the standard InfoICE loss used in contrastive learning settings.

Besides introducing the **relation-steering loss**, the authors introduce **relational-focal impor-**

**tance sampling**. To understand this, keep in mind that high-level semantics (like object relations) appear earlier in the denoising process, while finer details (like textures) emerge later. Since the goal of the model is to capture relations (a high-level concept) rather than details, focusing optimization on early stages of denoising (where high-level semantics appear) is crucial.

Normally, the timestep t in diffusion models is sampled uniformly from all possible timesteps. However, to emphasize high-level relations, the paper proposes a skewed sampling strategy where larger timesteps (closer to the initial noise, where high-level semantics are more prominent) are sampled with higher probability. This ensures that the model learns more about object relations rather than focusing on low-level pixel details.

The paper defines the sampling function:

$$f(t) = \frac{1}{T}\left(1 - \alpha \cos\left(\frac{\pi t}{T}\right)\right) \tag{72}$$

Where T is the total number of timesteps and $\alpha \rightarrow [0, 1]$ controls the skewness of the distribution. Thus, we have the following sampling function:

$$L_{\text{noise }} = \mathbb{E}_{t \sim f}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{x}_t, \tau_\theta(\vec{c})\right)\right\|^2\right] \tag{73}$$

This loss encourages the model to prioritize learning relations rather than focusing on reconstructing fine details. The final optimization objective of the ReVersion framework is a weighted combination of the steering loss (which guides the model towards learning object relations) and the denoising loss (which captures high-level semantics using importance sampling):

$$\langle R\rangle = \operatorname{argmin}_{\langle r\rangle}\left(\lambda_{\text{steer }} L_{\text{steer }} + \lambda_{\text{denoise }} L_{\text{denoise }}\right) \tag{74}$$

To understand the framework of reversion, we carry out an extensive example. Suppose the input text is "A cat is sitting on a chair." The relation words in this prompt are ['on', 'above', 'below'] and Stop Words are ['a', 'is', 'the', 'of', 'in', 'at', 'by']. We would use the following as the Placeholder Token *. Say we tokenize the input sentence "A cat is sitting on a chair.". After tokenization, we might get the token IDs assocaited with each word with the addition of [BOS] and [EOS] tokens. For our example, on is a relation word, a and is are stop words and other tokens like cat, sitting, and chair carry significant meaning. These tokens are converted into embeddings. Next, the authors create a stop mask that will identify the stop words, relation words, and special tokens. For our example, the words "cat", "red" and "chair" would be the negative samples in the contrastive loss calculations, stop-words would be removed. The positive samples are selected from the *relation_words* defined before hand. In this case, we might randomly sample one or more relation words, such as 'on', 'above', or 'below', and create embeddings for them.

### 3.5 Encoder-Based Fast-Tuning

The work [14] lives up to the promise that the authors made in their initial paper [13] of personalizing the diffusion model using a single image $\mathcal{J}$ rather than a collection of images. To understand their approach, let us first go through the normal personalization scheme. Usually, we employ the VQ-VAE encoder to map $\mathcal{J}$ to a latent $\vec{z}_0$ which is then sent to a U-NeT model that iteratively learns to map $\vec{z}_0$ to $\vec{z}_T$. At every point, the reconstruction is conditioned on the pseudovector $\langle\vec{d}\rangle$. The loss reads as following: $L_{LDM} = \mathbb{E}\left[\left\|\varepsilon - \varepsilon_\theta\left(\vec{z}_t, t, c_\theta\left(\langle\varepsilon_c\rangle\right)\right)\right\|^2\right]$

Now in this case, $\langle\vec{d}\rangle$ is generated by mapping the placeholder "*" to the CLIP space by tokenization and then continually improved by backpropagating using the above loss. In this paper, however, the authors first learn an encoder $E(\mathcal{J})$ that directly takes the generated image $\mathcal{J}$ to the clip-embedding space:

$$E(\mathcal{J}) \rightarrow \text{ CLIP-Embedding Space } \rightarrow \langle\vec{d}\rangle \tag{75}$$

Thus, encoder automatically map the image to the embedding space, providing an initial guess for the pseudovector $\langle\vec{d}\rangle$. With an encoder, we can bypass the iterative optimization process which traditionally requires multiple gradient updates to fine-tune the pseudovector. Thus, an encoder provides a **pre-learned mapping that reduces the number of required optimization steps**, or in some cases, removes the need for additional optimization entirely.

Therefore, if we can learn an encoder $E(\mathcal{J})$ that provides a good guess for $\langle\vec{d}\rangle$, we would have a significant speedup. Now, in order to make sense of what a good encoder should do, the authors explore what a good personalized embedding $\langle\vec{d}\rangle$ should look like. How well $\langle\vec{d}\rangle$ is optimized over depends on two things - distortion vs editability:

- If the latent code is too close to the real word embeddings (representing common objects or words), the concept might not be unique enough, leading to poor reconstruction.
- If the latent code is too far from the real word embeddings, the model might accurately reconstruct the concept but lose editability-meaning it will not generalize well to different contexts or prompts.

To provide an initial guess of $\langle\vec{d}\rangle$, the authors think in terms of a previous work in GANs. It suggests that the initial inversion constrains $\langle\vec{d}\rangle$ to an editable region of the latent space, at the cost of providing only an approximate match for the concept. We can then approximate that region by fine-tuning the generator. For example, suppose you have a novel cat "Rosy". We would first aim to map it to the "cat" category in the word-embedding, and only then would we optimize our diffusion model to approximate from "cat" to "Rosy":

$$\langle\vec{d}\rangle = \langle\vec{c}\rangle + s \cdot E\left(I_c\right) \tag{76}$$

Where E is our encoder, $\langle\vec{c}\rangle$ is the pre-trained model's embedding for the domain's coarse descriptor, and $s$ is a scaling factor which we empirically set to 0.1 . They further introduce a regularization term for the encoder:

$$\left\|E\left(I_c\right)\right\|_2^2 \tag{77}$$

The term $\left\|E\left(I_c\right)\right\|_2^2$ ensures that the encoder learns simpler, more generalizable representations of the concept, avoiding overfitting to specific details of the input image.

As for the encoder $E\left(I_c\right)$, the authors use CLIP visual encoder to directly map $I_c$. The encoder extracts feature from the pre-trained OpenCLIP ViT-H/14 model:

$$E\left(I_c\right) = f_{\text{clip }}\left(I_c\right) + \ldots \tag{78}$$

The ... indicates the need for us to incorporate how the encoder learn to estimate the noisy latents from $\vec{z}_0$ to $\vec{z}_T$. When denoising from $\vec{z}_T$ to $\vec{z}_0$, they argue that mapping the latent back to the image incurs a significant cost in both memory and time. Thus, instead of carrying that out, at layer of the U-NeT, they extract the pooled features $\left\{\vec{f_1}, \vec{f_2}, \ldots \vec{f_6}\right\}$. Thus, the encoder becomes:

$$E\left(I_c\right)=E\left(f_t^{(l)} \otimes f_{clip}\left(I_c\right)\right) \tag{79}$$

To understand [?], let us consider $f_{\text{clip }}\left(I_c\right)$ more closely. The ViT divides the images into patches, embed them, and process the sequence of patches using a transoformer model. Now, ViT has multiple layers, 12, 16, 24 depending on the model. Instead of taking features from each and every layer (which can be redundant or too detailed), the authors decided to extract features from every second layer. Furthermore, they only extract the [CLS token]. This means if you have a model with 12 layers, you would extract the [CLS] token features from layers 2, 4, 6, 8, 10, and 12.

$$F_l=CLIP_l\left(I_c\right)_{CLS}$$

Each extracted feature is passed through a linear layer:

$$h_l=W_l F_l+b_l$$

The transformed feature vectors $h_l$ are then averaged across all the extracted layers, which forms a hierarchical representation. If there are n layers being used, the pooled feature can be written as:

$$h_{\text{pooled }}=\frac{1}{n} \sum_{l=1}^n h_l$$

The pooled features are passed through a LeakyReLU activation function:

$$h_{\text{activated }}=\operatorname{LeakyReLU}\left(h_{\text{pooled }}\right)$$

After activation, the resulting features are passed through another linear layer to predict the embedding offset $E\left(\mathcal{J}_c\right)$ where $\mathcal{J}_c$ is the input concept image:

$$E\left(\mathcal{J}_c\right)=W_{\text{final }} h_{\text{activated }}+b_{\text{final }}+\ldots$$

Incorporating the features from the diffusion model, we have the final encoding off-set as following:

$$E\left(\mathcal{J}_c\right)=W_{\text{final }} h_{\text{activated }}+b_{\text{final }} \otimes f_t^{(l)} \tag{80}$$

In total, we have:

$$\langle\vec{d}\rangle_t=\langle\vec{c}\rangle+E\left(W_{\text{final }} h_{\text{activated }}+b_{\text{final }} \otimes f_t^{(l)}\right) \rightarrow \tag{16}$$

This word-embedding encoder predicts new code in the diffusion model's embedding space which best describes the input concept. Notice that during the time of training, this embedding is dynamic because of the features $f_t^{(l)}$ change at each iteration. Thus for each time step, we have $\langle\vec{d}\rangle_t$

**Weight-Set Optimization**

While the encoder is highly effective at creating a concept embedding, it doesn't directly control how the intermediate layers (especially the attention layers) of the diffusion model apply this new embedding to the image generation process. To mitigate these shortcomings, the authors introduce weight-offsets to optimize their model. Now, the authors carried out the normal textual inversion process, and they found that cross and self-attention layers undergo the highest change during tuning. Thus, these layers play the most crucial part in the tuning effort. Keeping this in mind, they modify three attention projection matrices - $W_Q, W_K$ and $W_v$. The weight offsets for the attention matrices are learned using an update rule:

$$W_{q,k,v}^{i} = W_{q,k,v,0}^{i} \cdot \left(1 + \Delta W_{q,k,v}^{i}\right) \tag{81}$$

Here, $W_{q,k,v,0}^{i}$ are the original pre-trained weights, and $\Delta W_{q,k,v}^{i}$ are the learned offsets. The goal is to learn the smallest possible perturbations to these attention weights that still enable the model to incorporate new concepts.

**Brief Overview of LoRA** In large-scale models, the parameter matrices in neural networks (such as weights in attention layers) are typically high-dimensional and dense. LoRA assumes that the updates required for fine-tuning the model can be captured in a low-rank subspace of the parameter matrices. The key idea is to restrict the updates to this low-rank space, thus greatly reducing the number of trainable parameters. Recall that **the rank of a matrix** is the number of linearly independent rows or columns it has. In LoRA, a low-rank update means that instead of allowing full-rank changes to the model's weight matrices, we approximate the update using a low-rank decomposition. Assume you have a weight matrix $W_0$ in the model that you want to update. Instead of directly fine-tuning $W_0$, LoRA introduces two smaller matrices, A , and B such that: $W = W_0 + \Delta W = W_0 +$ BA. In this representation, A is a low-rank matrix of dimension $r \times$ d where r is much small than original matrix. On the other hand, B is a low-rank matrix of dimension $d \times$ r where d is the original dimension of $W_0$

Instead of directly learning the offsets $\Delta W_{q,k,v}^{i}$ the offsets are regularized through a neural network to ensure smoothness and avoid overfitting:

$$\Delta W_{q,k,v}^{i} = \operatorname{Linear}\left(v_x \cdot v_y\right) \tag{82}$$

Where $v_x$ and $v_y$ are linear projections of a learned initial parameter vector $v_0$. During pre-training, the encoder and weight offsets are trained on large datasets using a combination of a diffusion loss and the regularization loss:

$$\begin{aligned} L &= L_{\text{diff }} + \lambda_r L_{reg} \\ L &= L_{\text{diff }} + \lambda_r \left\|E\left(I_c\right)\right\|_2^2 \end{aligned} \tag{83}$$

Where $\left\|E\left(I_c\right)\right\|_2^2 = E\left(W_{\text{final }} h_{\text{activated }} + b_{\text{final }} \otimes f_t^{(l)}\right)$

While the following method can achieve high-fidelity personalization with short training times, the work is not without limitations. First, the encoders rely on learning to generalize from large datasets that represent the coarse target class. This reliance means that the method is applicable only to classes where such large datasets exist. In practice, this includes categories such as faces and artistic styles, which are the primary use cases for personalization. However, this limitation restricts the method's applicability to rare or one-of-a-kind objects. For concepts within nearby domains—such as dogs personalized using a model trained on cats—the method still produces high-fidelity results. In contrast, for more distant domains, such as a wooden toy, the method fails to capture concept-specific details accurately.

A further limitation of this work lies in its requirement for inference-time tuning. Although the additional synthesis time is relatively minor, the approach requires that the inference machine be capable of performing model tuning. Moreover, the need to tune both the encoder and text-to-image models simultaneously increases the memory requirements compared to direct fine-tuning methods. These constraints may limit the method's usability in resource-constrained environments or applications with non-standard datasets.

# 4 Semantic Correspondence and Semantic Segmentation

## 4.1 TO-Do and Ideas

1. Can we use Diffusion models to make predictions on constitutionality? For example, given a ”leg”, classify whether the leg belongs to a 'human', 'cat', or a 'dog'. That is, the task of **using diffusion models to predict objects from their parts**
2. Using diffusion models for conditional generation where an existing image or video can be edited to modify the nature of the interaction (e.g., changing how a person interacts with an object). This could be useful in creative applications or even training simulations.
3. Write an intro explaining how self-attention naturally aids segmentation tasks and cross-attention aids correspondences task
4. I think I understand the theoretical reason behind why ”catastrophic forgetting” occurs. I was comparing the words in Diffusion Model's Clip tokenizer and playing with them and realized that for words like ”cat” and ”dog”, the similarity is 1 for the word-embedding. The Clip's Encoder space is unable to distinguish between the two
5. Incorporate more information in DiffewS

## 4.2 Introduction to Semantic Segmentation

**Semantic image segmentation** aims to assign a corresponding class label to every pixel in an image. Due to the pixel-level granularity of predictions, this task is often referred to as **dense prediction**. It is important to note that semantic segmentation does not differentiate between instances of the same class; it focuses solely on categorizing each pixel. For example, if an image contains two objects belonging to the same category, the resulting segmentation map does not inherently distinguish these as separate entities. This distinction is addressed by a different type of model, known as **instance segmentation**, which separates individual instances of the same class.

Segmentation models are instrumental in a wide range of applications including:

- **Autonomous Vehicles:** These models enable vehicles to perceive and interpret their surroundings, a critical capability for ensuring the safe integration of self-driving cars into existing road systems.
- **Medical Image Diagnostics:** Segmentation models enhance the efficiency and accuracy of radiological analysis, significantly reducing the time required for diagnostic assessments by augmenting the work of medical professionals.

**Representing the Task:** Given an input I of size W×H×C, the task is to produce a segmentation map S of the same size where each pixel $S(x, y)$ corresponds to a class label c from a set of predefined classes C = $\{c_1, c_2 \dots c_N\}$ where N is the number of classes. Then, the segmentation map S can be defined as:

$$\mathrm{S} : \{1, 2 \dots \ \mathrm{W}\} \times \{1, 2, \dots \mathrm{H}\} \rightarrow \{\mathrm{c}_1, \mathrm{c}_2, \dots \mathrm{c}_\mathrm{N}\} \tag{84}$$

For each pixel (x, y) in the image $I$ , the task is to predict the class label $S(x, y)$ that best describes the object or region in which the pixel belongs. To understand the framework, imagine we have a simple $4 \times 4$ pixel image and our task is to perform semantic segmentation on it for a scenario where we have distinct classes: Background ($c_1$), Cat ($c_2$) and Dog ($c_3$). For simplicity, lets consider the following ( $4 \times 4$ ) image representation where each letter represents a pixel belonging to a different class:

$$\text{Ground Truth Segmentation Map } \mathrm{S}^*(\mathrm{x}, \mathrm{y}) = \begin{bmatrix} B & B & C & C \\ B & D & D & C \\ B & D & D & C \\ B & B & C & C \end{bmatrix}$$

Where B represents the Background, C represents the Cat and D represents the Dog. We call S* as the ground truth segmentation map. The goal of semantic segmentation is to assign each pixel in the image a label corresponding to one of these classes. After processing this image through a semantic segmentation model, we aim to get a segmentation map $S$ that mirrors the layout of the ground truth segmentation map in terms of class assignments. The segmentation map S for the given example could be represented as a matrix of the same dimension as the image, where each element is the class label predicted for the corresponding pixel:

$$\text{Segmentation map } S(x, y) = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 3 & 3 & 2 \\ 1 & 3 & 3 & 2 \\ 1 & 1 & 2 & 2 \end{bmatrix}$$

Here, the numbers (1, 2, 3) correspond to Background, Cat and Dog. So $S(x, y)$ gives us the class of the pixel at position ( x, y ) in the image. After we have the segmentation map S(x, y ), we compare it with the ground truth segmentation map $\mathrm{S}^*$. A common choice is the cross-entropy loss. For a single pixel, it is defined as following:

$$L_{CE}\left(S(x, y), S^*(x, y)\right) = -\log\left(\frac{\exp\left(S(x, y)\left[c^*\right]\right)}{\sum_{c \in C} \exp(S(x, y)[c])}\right) \tag{85}$$

Where $S(x, y)[c]$ is the predicted probability of pixel (x, y) belonging to class c , and $\mathrm{c}^*$ is the actual class label for that pixel in the ground truth segmentation map $S^*$. The total loss for the image is the sum of the pixel-wise losses:

$$L = \sum_{x=1}^{W} \sum_{y=1}^{H} L_{CE}\left(S(x, y), S^*(x, y)\right) \tag{86}$$

Note that the above approach assumes one-hot encoding of class labels. For example, consider the image above. The total classes we have are {person, purse, sidewalk, building.

Equation 86 highlights that training a semantic segmentation model can be computationally expensive when considering pixel-wise loss since the loss is calculated and summed over all pixels in the image for each training instance. One way to mitigate this is through a method know as the **"center pixel prediction"**. In this approach, instead of predicting the class label for each pixel individually, the model focuses on predicting the class label for the center pixel of a small region or patch around each pixel. This center pixel is typically chosen to be the pixel at the center of the receptive field of the convolutional neural network (CNN). The predicted class label for this center pixel is then assigned to all pixels in the corresponding patch.

The cross-entropy loss calculates predictions for each pixel vector independently and averages the results across all pixels. This approach essentially assigns equal importance to each pixel during training. However, if there is an imbalance in class representation within an image, this method can lead to biased training dominated by the most common class. To address this issue, Long et al. (FCN paper) propose weighting the loss for each output channel to balance class representation in the dataset.

Ronneberger et al. (U-Net paper) introduce a different strategy that adjusts the loss weight for each pixel. Their method assigns greater weight to pixels at the borders of segmented objects, which helps their U-Net model achieve better segmentation of cells in biomedical images. This technique allows individual cells to be easily identified in binary segmentation maps, even when the cells are closely packed.

Another widely used loss function for image segmentation tasks is derived from the Dice coefficient, a metric that measures the overlap between two sets. The Dice coefficient ranges from 0 to 1, with a value of 1 indicating complete overlap. Originally developed for binary data, the Dice coefficient can be computed as:

$$\text{Dice} = \frac{2|A \cap B|}{|A| + |B|} \tag{87}$$

**Encoder-Decoder Structures for Segmentation**: As discussed, the process of associating pixel-level information with image patches can be computationally expensive. To address this challenge, convolutional neural networks are often employed. Although a simple convolutional neural network can be used for segmentation, the computational burden of pixel-level operations remains substantial. To reduce this burden, one widely adopted approach for image segmentation involves utilizing an **encoder-decoder architecture**. In this design, the encoder module reduces the spatial resolution of the input image, creating low-resolution feature mappings that efficiently capture discriminative information for classification. The decoder module then upsamples these feature representations to reconstruct a full-resolution segmentation map. In CNN architectures, transposed convolutions are utilized.

A limitation of encoder-decoder structure is that the encoder module significantly reduces the spatial resolution of the input, making it challenging for the decoder module to produce fine-grained segmentations. This issue highlights a fundamental trade-off between semantic and spatial information. While global features resolve *what* is present in the image, local features resolve *where* objects are located. To address this, the authors of Ronneberger et al. introduced a technique that gradually upsamples the encoded representation in stages, incorporating **skip connections** from earlier layers. These skip connections merge fine-grained features from the encoder with coarse-grained features from the decoder. This enables the model to make localized predictions that align with global context.

**Zero-shot semantic segmentation:** Let the class space be $C = C_{\text{seen}} \cup C_{\text{unseen}}$ where $C_{\text{seen}}$ are the classes seen during training and $C_{\text{unseen}}$ are classes unseen during training. The auxiliary information $A(c)$ for each class $c$ provides semantic representations that link seen and unseen classes. Formally, the task is to predict the pixel-wise segmentation mask $\hat{M}$ for an image $I$, assigning labels to pixels in $C_{\text{unseen}}$, without directly observing labeled data for $C_{\text{unseen}}$ during training.

The methodology often utilizes a feature extractor and additionally, there is a function $g(c)$ which encodes the auxiliary information of each class $c$. Given these, we define the **Compatibility Function** $\phi(F, g(c))$, which measures the compatibility between image features and class embeddings, often defined as:

$$\phi(F, g(c)) = F(x, y) \cdot g(c)$$

During inference, the segmentation mask $\hat{M}$ is predicted by assigning each pixel the label of the most compatible class:

$$\hat{M}(x, y) = \underset{c \in C_{\text{unseen}}}{\text{argmax}}\ \phi(F(x, y), g(c))$$

where $F(x, y)$ represents the feature at pixel location $(x, y)$.

A simple example of zero-shot segmentation is segmenting animals in an image where the model was trained to recognize only cats and dogs but is tasked with identifying a horse (an unseen class). The model uses semantic information like textual descriptions (e.g., ”a horse has four legs and a tail”) to generalize and segment the horse in the image without ever having seen labeled examples of horses during training

**Few-shot semantic segmentation:** Few-shot segmentation involves segmenting objects in an image where only a few annotated examples (support samples) of the target class are available for training. Let the support set be $S = (I_s^i, M_s^i)_{i=1}^K$, where $I_s^i$ is the $i$-th support image, and $M_s^i$ is its corresponding segmentation mask for the target class. Given a query image $I_q$, the goal is to predict a segmentation mask $M_q$ that identifies regions in $I_q$ corresponding to the target class in $S$.

The methodology utilizes a feature extractor for both support and query images. A **Support-Query Relationship Function**

$$\phi(S, I_q)$$

is defined to measure the relationship between the support set and the query image, allowing the model to adapt to the target class with limited examples.

A simple example of few-shot segmentation is segmenting a dog in a query image when only one support image of a dog with its segmentation mask is provided. The model learns from this single annotated example to identify and segment the dog in the query image.

**Unsupervised semantic segmentation:** Unsupervised semantic segmentation involves segmenting an image into semantically meaningful regions without the use of any labeled data. Let an input image be $I$, and the goal is to predict a segmentation mask $\hat{M}$, where each pixel is assigned to a cluster that represents a distinct semantic region, without any prior supervision or annotations.

The methodology often employs unsupervised feature extraction techniques, such as clustering in the feature space, to group pixels with similar semantic meanings. A **Clustering Function** $\phi(F)$ is used, where $F$ represents the extracted features from the input image $I$, to partition the image into clusters corresponding to semantic regions.

A simple example of unsupervised semantic segmentation is segmenting an image of a natural scene into regions such as sky, water, and land, without any labeled data. The model clusters pixels with similar visual patterns, such as blue for the sky and water or green for the land, to generate a segmentation mask.

## 4.3 Segmentation using Statistical Physics

There are a few ideas from statistical physics that can be utilized for segmentation purposes. Both the **Ising model** and the **Potts model** can be used for image segmentation. The Potts model is often used because it allows for multiple possible labels (or states) at each pixel, corresponding to different segments in the image. The Ising model can also be used for binary segmentation tasks (where there are only two possible labels, such as "foreground" and "background").

**Ising Model for Binary Segmentation:** In the Ising model, each site (or pixel in the case of image segmentation) can take one of two possible states, typically denoted as $s_i \in \{-1, +1\}$. The energy function for the Ising Model is:

$$H(s) = -J \sum_{<i,j>} s_i s_i - \sum_i h_i s_i \tag{88}$$

Where $h_i$ is the local external field acting on pixel $i$. We can think of the terms in equation (4) as consisting of the following two terms:

1. Unary term $\sum_i h_i s_i$ : This term represents how well each pixel fits into a particular class (based on the pixel intensity or other features).
2. Pairwise term $\sum_{<i,j>} s_i s_i$ : This term encodes the interaction between neighboring pixels, penalizing configurations where neighboring pixels have different labels (to enforce smoothness).

In binary segmentation, the goal is to segment the image into two regions, $\mathcal{R}_1$ (foreground) and $\mathcal{R}_2$ (background). The Ising model provides a natural framework for this problem by associating each pixel with a spin variable $s_i$ where,

- $s_i = +1$ represents that pixel i belongs to the foreground
- $s_j = +1$ represents that pixel j belongs to the foreground

The segmentation task can be cast as finding the configuration of spins $s = \{s_1, s_2 \ldots, s_N\}$ that minimizes the following energy function:

$$E(s)=\sum_i \psi_u\left(s_i, I_i\right)+\sum_{<i, j>} \psi_p\left(s_i, s_j\right)$$

Where $\psi_u\left(s_i, I_i\right)$ is the unary potential (or data term) that encodes how well pixel i's intensity $I_i$ fits with label $s_i$ (foreground or background). $\psi_p\left(s_i, s_j\right)$ is the pairwise potential that encodes the interaction between neighboring pixels $i$ and $j$, encouraging neighboring pixels to have the same label.

The unary potential is typically derived from the likelihood of pixel i's intensity given the foreground or background distribution. Suppose the pixel intensities are modeled as Gaussian distributions for the foreground and background:

$$P\left(I_i \mid s_i=+1\right)=\mathcal{N}\left(I_i ; \mu_1, \sigma_1^2\right), P\left(I_i \mid s_i=-1\right)=\mathcal{N}\left(I_i ; \mu_2, \sigma_2^2\right)$$

The unary term $\psi_u\left(s_i, I_i\right)$ can then be defined as the negative log-likelihood:

$$\psi_u\left(s_i, I_i\right)=\left\{\begin{array}{ll}-\log P\left(I_i \mid s_i=+1\right) & =\frac{\left(I_i-\mu_1\right)^2}{2 \sigma_1^2}, \text { if } s_i=+1, \\ -\log P\left(I_i \mid s_i=-1\right) & =\frac{\left(I_i-\mu_2\right)^2}{2 \sigma_2^2}, \text { if } s_i=-1\end{array}\right. \tag{89}$$

This unary term encourages pixels with intensities close to $\mu_1$ to be labeled as foreground and those close to $\mu_2$ to be labeled as background $s_i=-1$

The pairwise potential penalizes neighboring pixels that have different labels (i.e., different spin states) to enforce spatial smoothness. The simplest form of the pairwise term is:

$$\psi_p\left(s_i, s_j\right)=J s_i s_j$$

Where J > 0 encourages neighboring pixels i and j to have the same label (spin state). This term is inspired by the original Ising model, where spins prefer to align with their neighbors to minimize the energy.

The total energy is:

$$E(s)=\sum_i \psi_u\left(s_i, I_i\right)+\sum_{<i, j>} J s_i s_j \tag{90}$$

The goal is to minimize the energy $E(s)$ to find the optimal segmentation. This can be a challenging optimization problem because the energy function involves interactions between neighboring pixels (pairwise terms).

**Potts Model for Segmentation** : The Potts model extends the Ising model for multi-class segmentation. The energy function for the Potts model is defined as:

$$E(Y)=\sum_{i, j} \mathbb{I}\left(Y_i \neq Y_j\right) \tag{91}$$

Where $Y_i$ and $Y_j$ are the labels of neighboring sites and $\mathbb{I}\left(Y_i \neq Y_j\right)$ is an indicator function that penalizes neighboring sites if they have different labels. In some sense, the Potts model is the generalization of the Ising Model. The Ising model energy is defined as following in case of no external magnetic field:

$$H=-J \sum_{\langle i, j\rangle} s_i s_j \tag{92}$$

Where $s_i \in\{-1,+1\}$ and we know that the Ising model prefers configurations where the spins are aligned. The Potts model generalizes the Ising model by allowing more than two possible spin states at each site. It is used to model systems where each site (or particle) can be in one of **q** discrete states. The Hamiltonian (energy function) for the Potts model is given by:

$$H=-J \sum_{\langle i, j\rangle} \delta\left(s_i, s_j\right)$$

For q = 2, the Potts model is equivalent to the Ising model but for $q>2$, the Potts model can exhibit a first-order (discontinuous) phase transition, where the transition between ordered and disordered states occurs abruptly.

**Graph Optimization for Solving the Ising Hamiltonian:** The Ising Model (90) is solved using graph optimization. To carry this out, we first create a graph in which each pixel i in the image corresponds to a node in the graph. Two additional nodes are added: the source node S (representing the foreground) and the sink node T (representing the background).

**Insert Figure Here!**

There are two types of edges in the graph:

1. **T-Links**: These connect each pixel node i to either the source node S or the sink node T . The weight of these edges encode the unary terms $\psi_u\left(s_i\right)$ which represents the cost of assigning pixel $i$ to the foreground or background. For each pixel i, two edges are added:
   (a) An edge from i to the source with weight $w(i, S)=\psi_u\left(s_i=1\right)$ (cost of assigning i to the foreground)
   (b) An edge from i to the sink with weight $w(i, T)=\psi_u\left(s_i=0\right)$ (cost of assigning i to the background)
2. **N-links**: These connect neighboring pixel nodes i and j in the terms. The weights of these edges encode the pairwise terms $\psi_p\left(s_i, s_j\right)$ which represent the cost of assigning different labels to neighboring pixels.

The goal of the segmentation problem is to minimize the energy function $E(s)$ by cutting the graph in such a way that the total cost (or energy) is minimized. The segmentation problem is solved by finding the minimum cut that separates the graph into two disjoint sets:

- One set containing the source node $S$ and the pixels labeled as foreground.
- The other set containing the sink node T and the pixels labeled as background.

The cut is defined as a set of edges that, when removed, separates the source from the sink. The cost of the cut corresponds to the sum of the weights of the edges in the cut. More formally, Given a graph G = (V, E) where V is the set of vertices including the source S and sink T and E
is the set of edges with associated weights, a cut C in the graph is a partition of the vertices into two disjoint sets:

- S $\subseteq$ V containing the source.
- T $\subseteq$ V containing the sink.

The cost of the cut $C$ is defined as the sum of the weights of the edges that cross the cut:

$$\operatorname{cost}(C)=\sum_{(i,j)\in C} w(i,j) \tag{93}$$

The minimum cut is the cut that has the smallest possible cost, i.e., the one that minimizes the total energy function $E(s)$ of the segmentation.

Unary Term Representation: The unary term $\psi_u\left(s_i\right)$ assigns each pixel to either the foreground or background based on the likelihood that the pixel belongs to that class (e.g., using intensity or color information). This is represented as the capacity of the T-links:

- $\mathrm{w(i,S)}=$ cost of assigning pixel i to the foreground
- $\mathrm{w(i,T)}=$ cost of assigning pixel $i$ to the background

Pairwise Term Representation: The pairwise term $\psi_p\left(s_i, s_j\right)$ encourages smoothness by penalizing neighboring pixels with different labels. This is represented as the capacity of the N links between neighboring pixels:

$$\mathrm{w(i,j)}=\text{ cost of assigning different labels to neighboring pixels i and}$$

This weight is often modeled as:

$$w(i,j)=\lambda \exp\left(-\frac{\left\|I_i-I_j\right\|^2}{2\sigma^2}\right) \tag{94}$$

Where $\left\|I_i-I_j\right\|^2$ is the squared difference in intensity or color between pixels i and j , and $\lambda$ controls the strength of the smoothness penalty. The Gaussian term ensures that neighboring pixels with similar intensities are more likely to be assigned the same label.

## 4.4 PACL

The paper "Open Vocabulary Semantic Segmentation with Patch Aligned Contrastive Learning" [25] introduces a method called Patch Aligned Contrastive Learning (PACL) for aligning image patches from a vision encoder with text tokens from a text encoder in models like CLIP. The approach is designed for **zero-shot semantic segmentation**, meaning it can segment images into classes not explicitly present in its training data without requiring segmentation annotations.

In traditional contrastive learning, such as in models like CLIP, the objective is to learn a similarity between entire images and their corresponding text descriptions, globally aligning the CLS token (a special token summarizing the entire image or text) from the image encoder and text encoder. This alignment helps the model understand whether a given image matches a description, but it works at the level of whole images, not specific parts of images (patches).

CLIP uses a contrastive loss to align image-text pairs. Each image and its corresponding text are represented by a global embedding. Specifically, the image encoder $f_v$ takes an image x and generates a CLS token, a single embedding representing the entire image:

$$f_v(x) \in \mathbb{R}^{D_v} \tag{95}$$

Where $D_v$ is the dimension of the embedding.
In PACL (Patch Aligned Contrastive Learning), the goal is to move beyond this global alignment and perform a finer, patch-level alignment. Here, instead of aligning the entire image and text globally, PACL aligns individual image patches (parts of the image) with the CLS token from the text encoder. This allows the model to identify which parts of the image correspond to specific text descriptions.

Instead of encoding the whole image into a single CLS token, the vision encoder in PACL breaks the image into patches and outputs a series of embeddings, one for each patch. So, instead of $f_v(x) \in \mathbb{R}^{D_v}$, PACL uses:

$$f_v(x) \in \mathbb{R}^{TxD_v} \tag{96}$$

Where T is the number of patches in the image. The text encoder still produces a global CLS token for the entire text y. PACL computes the cosine similarity between the text CLS token and each of the T patch embeddings. This gives T similarity values, one for each patch:

$$s(x, y) \in \mathbb{R}^T, s(x, y)_i = \frac{e_v\left(f_v(x)_i\right) \cdot e_t\left(f_t(y)\right)}{\left|e_v\left(f_v(x)_i\right)\right|\left|e_t\left(f_t(y)\right)\right|} \tag{97}$$

Once we have the similarity scores $s(x, y)_i$ for all patches in the image, PACL uses these scores to create weights for each patch. The weights are computed using the softmax function over all the patches:

$$a(x, y)_i = \frac{\exp\left(s(x, y)_i\right)}{\sum_{j=1}^{T} \exp\left(s(x, y)_j\right)} \tag{98}$$

To generate a global image representation, PACL aggregates the patch embeddings by taking a weighted sum where the weights are derived from the patch-text similarity scores $s(x, y)$ :

$$\hat{v} = \sum_{i=1}^{T} a(x, y)_i e_v\left(f_v(x)_i\right) \tag{99}$$

By summing the weighted embeddings of all patches, we get a single vector $\hat{v}$ that represents the most relevant parts of the image, considering the text description. In other words, instead of treating each patch equally, the model emphasizes the patches that are most relevant to the text, aggregating them into a single vector.

During training, PACL uses the weighted patch embedding $\hat{v}$ and aligns it with the text CLS token using a contrastive loss (InfoNCE). The training objective is to make sure that the weighted sum of patches from the correct image-text pair is more similar than any incorrect pair.

The InfoNCE loss is used, where for each correct image-text pair $(x_i, y_i)$, the similarity $\hat{\phi}(x, y)$ should be maximized, while the similarity for incorrect pairs $\hat{\phi}\left(x_i, y_i\right)$ should be minimized:

$$\widehat{\phi}(x, y) = \frac{\hat{v} \cdot e_t\left(f_t(x)_i\right)}{|\hat{v}| \cdot \left|e_t\left(f_t(y)\right)\right|} \tag{100}$$

Thus, the $L_{\text{InfonCE}}$ loss assumes the form:

$$L_{\text{InfoNCE}} \ = -\log \frac{\exp\left(\widehat{\phi}\left(x_i, y_i\right)\right)}{\sum_{j=1}^{k} \exp\left(\widehat{\phi}\left(x_j, y_j\right)\right)} \tag{101}$$

This contrastive loss encourages the model to learn the alignment between the relevant image patches and the text, resulting in a strong patch-text correspondence. At inference time, PACL can be used for zero-shot semantic segmentation. Given an image and a set of class labels (in text form), the model computes the similarity between each patch in the image and each class label. For each class, a segmentation mask is generated based on the similarity scores between the patches and the class text.

**Example:** Imagine you have an image of a park. The image has multiple objects: trees, people, and a dog. The text description could be something like: " A dog running in the park". You want the model to learn that the part of the image containing the dog corresponds to the word "dog" in the text, and the part of the image containing the trees corresponds to "park.". The first thing PACL does is divide the input image into patches. Let's assume the image is divided into 16 patches (though in practice, there are usually more). Each patch is a small portion of the image, say:

- Patch 1: Part of the sky
- Patch 2: Part of a tree
- Patch 3: Part of a dog
- ... and so on

So we have 16 patches $x_1, x_2 \ldots x_{16}$. Next, the image patches and the text description are encoded by the model. The vision encoder $f_v$ processes each image patch $x_i$ and generates an embedding (a vector) for each patch. So, you get 16 vectors, one for each patch:

$$f_v\left(x_i\right) \in \mathbb{R}^{D_v}, \text{ for each patch, i} = 1, 2, \ldots 16$$

The text encoder $f_t$ processes the text ("A dog running in the park") and generates a CLS token embedding, which is a vector summarizing the entire text:

$$f_t(y) \in \mathbb{R}^{D_t}$$

The key is that these embeddings for the image patches and the text are projected into the same shared space (through functions $e_v$ and $e_t$ ) so they can be compared. Note that these functions $e_v$ and $e_t$ are learned during training. Using these learned functions, we learn:

$$s(x, y)_i = \frac{e_v\left(f_v(x)_i\right) \cdot e_t\left(f_t(y)\right)}{\left|e_v\left(f_v(x)_i\right)\right| \left|e_t\left(f_t(y)\right)\right|}$$

This gives a similarity score $s(x, y)_i$ for each patch. For example, for the text description "A person is walking with a dog in the park." And 4 patches: 1. Sky, 2. A tree, 3. A person and 4. A dog, the goal is to compute how much each patch relates to the given text description, using the following steps:

- $s(x, y)_1$ (Sky and text): Suppose this patch of sky is not relevant to the text "A person is walking with a dog in the park." So, the similarity score might be low, e.g., $s(x, y)_1 = 0.1$.

- $s(x, y)_2$ (Tree and text): A tree in the park is somewhat relevant to the text, but it's not the main focus. So, the similarity score might be moderate, e.g., $s(x, y)_2 = 0.4$

- $s(x, y)_3$ (Person and text): Since the text describes a person walking, this patch should be highly relevant. Therefore, the similarity score might be high, e.g., $s(x, y)_3 = 0.8$

- $s(x, y)_4$ (Dog and text): The dog is central to the text description, so this patch will have a high similarity score, e.g., $s(x, y)_4 = 0.9$.

The final softmax weights $a(x, y)_i = \frac{\exp(s(x,y)_i)}{\sum_{j=1}^{T} \exp(s(x,y)_j)}$ represent how much attention or weight each patch should get in the final representation. It comes out as $[0.152, 0.205, 0.306, 0.338] =$ [Sky, Tree, Person, Dog]. Thus, the two most relevant features of interest comes out to be dog and person. The final step is to compute the weighted sum of the patch embeddings using these weights to give $\hat{v}$. This is a new vector that represents the image, but with more emphasis on the patches that are most relevant to the text (in this case, the dog and the person).

The model then uses contrastive learning to align this vector $\hat{v}$ with the text CLS embedding $e_t\left(f_t(y)\right)$.

Inference: This involves using the learned model to perform tasks like zero-shot semantic segmentation without explicit training for segmentation. The model is able to infer which parts of the image correspond to specific text labels (like "dog", "person") by aligning image patches with text descriptions. During inference, we are given:

- An image (e.g., a photo of a park with a dog, a person, and trees).

- A set of class labels in the form of text (e.g., ["dog", "person", "tree"]). These labels might describe objects you want to segment or recognize in the image.

The task is to predict which parts of the image correspond to each label, producing a segmentation mask or classifying each patch with the most likely label. As in training, the first step is to divide the input image into patches. Let's assume the image is divided into T patches, each patch representing a small portion of the image:

- Patch 1: Part of the sky.

- Patch 2: Part of a tree.

- Patch 3: Part of the person.

- Patch 4: Part of the dog.

- And so on, for all T patches in the image.

Each patch is passed through the vision encoder $f_v$ to generate a feature embedding for each patch. The vision encoder produces T embeddings, one for each patch: $f_v\left(x_i\right) \in \mathbb{R}^{D_v}$ where $D_v$ is the dimensionality of the patch embeddings. The set of text labels is passed through the text encoder $f_t$. Each label (e.g., "dog", "person", "tree") is encoded into an embedding $f_t\left(y_c\right) \in \mathbb{R}^{D_t}$. For each patch i in the image and each class label $y_c$, we compute the cosine similarity between the patch embedding $f_v(x)_i$ and the text embedding $f_t\left(y_c\right)$. This similarity tells us how well the patch corresponds to the class label. The similarity $s\left(x, y_c\right)_i$ for patch i and class $y_c$ is computed as:

$$s(x,y)_i = \frac{e_v\left(f_v(x)_i\right) \cdot e_t\left(f_t(y)\right)}{\left|e_v\left(f_v(x)_i\right)\right|\left|e_t\left(f_t(y)\right)\right|}$$

This produces a similarity score between each image patch and each class label. For example, if patch 4 contains the dog and the text label is "dog", the similarity score $s(x, \mathrm{dog})_4$ would be high. Conversely, if patch 1 is part of the sky and the label is "dog", the similarity score $s(x, dog)_1$ would be low. For each patch, you compute a softmax over the class label similarity scores to assign each patch to the most likely class. The softmax is computed over all class labels C for each patch $a(x,y)_i = \frac{\exp(s(x,y)_i)}{\sum_{c=1}^{C}\exp(s(x,y)_j)}$. This softmax function converts the raw similarity

Once the softmax scores $a(x,y)_i$ are computed for all patches and class labels, each patch is assigned to the class with the highest probability. This can be used to generate a segmentation mask where each patch is labeled with the class that it most likely corresponds to.

For example:

- Patch 4 might be labeled "dog" $a(x,y)_4$ is high
- Patch 3 might be labeled "person".
- Patch 2 might be labeled "tree".

The resulting output is a segmentation map that tells you which parts of the image correspond to each object class.

### 4.5 PiCIE

The paper "PiCIE: Unsupervised Semantic Segmentation Using Invariance and Equivariance in Clustering" [10] presents a method for unsupervised semantic segmentation without the need for labeled data. The goal of this method is to discover and segment out high-level concepts (e.g., trees, sky, houses) from images without any human-provided annotations.

The key challenge in unsupervised semantic segmentation is how to cluster and label pixels in a way that reflects meaningful semantic objects. Traditional clustering methods are limited to simple, object-centric images, and fail when applied to complex, scene-centric datasets. PiCIE addresses this limitation by proposing a method that assigns cluster memberships to pixels while learning features that incorporate both invariance to photometric transformations (e.g., color changes) and equivariance to geometric transformations (e.g., cropping, flipping).

**Methodology:** PiCIE clusters pixels based on their feature representations. Let $\vec{x}_i$ represent an image from a dataset, and let $f_\theta\left(\vec{x}_i\right)$ be the feature representation of the image learned by a convolutional neural network ( CNN ) with parameters $\theta$. For each pixel p in the image, the CNN produces a feature vector:

$$f_\theta\left(\vec{x}_i\right) = \text{ Feature map for all pixels in the image}$$

PiCIE employs k-means clustering to group pixels based on their feature representations. Clustering is performed to discover semantic groups of pixels:

$$\min_{y,\mu} \sum_{i,p} \left\|f_\theta\left(x_i\right)[p] - \mu_{ip}\right\|^2$$

Where $y_{ip}$ is the cluster label of pixel p in image $x_i$ and $\mu_k$ is the centroid of the k -th cluster. The distance is typically the Euclidean distance in feature space.

The learning process is iterative and follows the standard k-means algorithm steps. Initially, the centroids $\mu_k$ for k $= 1, \ldots$ K are randomly initialized. For each pixel p in image $x_i$, its feature vector $f_\theta\left(x_i\right)[p]$ is compared to the current centroid $\mu_k$. Each pixel is assigned to the cluster whose centroid is closest in terms of the distance metric (typically Euclidean or cosine distance). The assignment step is given by $y_{ip} = \arg\min_k \left\|f_\theta\left(x_i\right)[p] - \mu_{ip}\right\|^2$. This assigns pixel p to the cluster whose centroid minimizes the distance. Once all pixels have been assigned to clusters, the centroids $\mu_k$ are updated by taking the mean of the feature vectors of all the pixels assigned to each cluster. The centroid update rule is $\mu_k = \frac{1}{\left|C_k\right|} \sum_{(i,p) \in C_k} f_\theta\left(x_i\right)[p]$ where $C_k$ is the set of pixels assigned to cluster $k$. After each iteration, the assignments of pixels to clusters and the positions of the centroids are updated. The process converges when the assignments of pixels to clusters no longer change significantly, or after a predefined number of iterations.

The paper titled "Semantic Correspondence as an Optimal Transport Problem" introduces a novel approach for establishing dense correspondences between semantically similar images. The authors address two main challenges in semantic correspondence: the many-to-one matching problem and the background matching problem. Given feature maps $f_s \in \mathbb{R}^{h_s \times w_s \times d}$ and $f_t \in \mathbb{R}^{h_t \times w_s \times d}$, the standard approach is to compute a correlation map C between these features using cosine similarity:

$$C_{ijkl} = \frac{\left(f_s\right)_{ij} \cdot \left(f_t\right)_{kl}}{\left\|\left(f_s\right)_{ij}\right\| \left\|\left(f_t\right)_{kl}\right\|} \tag{102}$$

Where $C_{ijkl}$ represents the similarity score between the pixel at position $(i,j)$ in the source image and the pixel at position $(k,l)$ in the target image. The individual matching approach typically assigns the best match for each source pixel $(i,j)$ by maximizing $C_{ijkl}$ over $(k,l)$ :

$$\left(k^*, l^*\right) = \operatorname{argmax} C_{ijkl} \tag{103}$$

However, this can result in many-to-one matching, where multiple source pixels match to the same target pixel, leading to suboptimal or ambiguous correspondences.

**PiCIE uses a non-parametric classifier:**: In a non-parametric classifier, there is no explicitly learned decision boundary or parametric model (like a neural network classifier). Instead, the classification is performed based on comparisons between data points and reference points (in this case, cluster centroids). The key idea is that the class of a pixel is determined by the similarity of its feature representation to the cluster centroids, without explicitly learning a classifier.

Once the centroids $\mu_k$ are computed and assigned to pixels, the non-parametric classifier is responsible for assigning pixel labels during training. Instead of learning a classifier function (like a fully connected layer with weights) to predict the label of each pixel, PiCIE uses the distance between the pixel feature vector and the cluster centroids to compute a softmax probability that assigns the pixel to a cluster (label).

This is done by computing the cosine similarity between the pixel feature $f_\theta(x)[p]$ and each cluster centroid $\mu_k$, followed by a softmax function to convert these similarities into probabilities. The pixel is then assigned to the cluster with the highest probability. The classification loss is based on the softmax cross-entropy between the pixel feature and the cluster centroids:

$$L_{\text{clust }}\left(f_\theta\left(\vec{x}_i\right)[p], y_{ip}, \mu\right)=-\log \left(\frac{\exp \left(-d\left(f_\theta\left(\vec{x}_i\right)[p]\right), \mu_{y_{ip}}\right)}{\sum_l \exp \left(-d\left(f_\theta\left(\vec{x}_i\right)[p]\right), \mu_l\right)}\right) \tag{104}$$

Here, d(·, ·) is the cosine distance between the feature and the centroid.

**Invariance to Photometric Transformations**: To make the clustering more robust, PiCIE introduces invariance to photometric transformations. Photometric transformations refer to changes in color, brightness, contrast, or other visual properties that do not alter the object structure. The goal is that the cluster assignments should not change under such transformations.

To enforce this, PiCIE generates two transformed versions of each image, $P^{(1)}\left(\vec{x}_i\right)$ and $P^{(2)}\left(\vec{x}_i\right)$ where P represents a random photometric transformation. For each pixel ppp, PiCIE computes the features under both transformations:

$$z_{ip}^1=f_\theta\left(P^{(1)}\left(\vec{x}_i\right)\right)[p], z_{ip}^2=f_\theta\left(P^{(2)}\left(\vec{x}_i\right)\right)[p] \tag{105}$$

PiCIE then performs k-means clustering on these two sets of feature vectors separately, resulting in two sets of cluster memberships and centroids, $\left(y^1, \mu^1\right)$ and $\left(y^2, \mu^2\right)$. The within-view loss encourages consistency within each view:

$$L_{\text{within }}=\sum_{i,p} L_{\text{clust }}\left(z_{ip}^1, y_{ip}^1, \mu^1\right)+L_{\text{clust }}\left(z_{ip}^2, y_{ip}^1, \mu^1\right) \tag{106}$$

The cross-view loss encourages the network to produce similar cluster assignments across different photometric transformations:

$$L_{\text{cross }}=\sum_{i,p} L_{\text{clust }}\left(z_{ip}^1, y_{ip}^2, \mu^2\right)+L_{\text{clust }}\left(z_{ip}^2, y_{ip}^1, \mu^1\right) \tag{107}$$

This forces the network to learn features that are robust to changes in photometric properties.

**Equivariance to Geometric Transformations**: PiCIE also enforces equivariance to geometric transformations (e.g., scaling, cropping, rotation). Unlike photometric transformations, geometric transformations change the spatial arrangement of the image, and the labels should change accordingly. For geometric transformations, PiCIE applies a random geometric transformation $G_1$ (such as random cropping or flipping) to the image. The feature vectors are computed as follows:

$$z_{ip}^1=f_\theta\left(G^{(1)} f_\theta\left(P^1\left(\vec{x}_i\right)\right)\right)[p], z_{ip}^2=G^{(2)} f_\theta\left(P^2\left(\vec{x}_i\right)\right)[p] \tag{108}$$

The geometric transformation $G_1$ is applied both to the image and the feature maps. This ensures that the pixel labels in the transformed image correspond to the labels of the original image, up to the geometric transformation. The final loss is a combination of the within-view and cross-view losses:

$$L_{\text{total }}=L_{\text{within }}+L_{\text{cross }}$$

This ensures that the network learns features that are invariant to photometric changes but equivariant to geometric transformations.

**Overclustering:** To improve the stability of clustering, PiCIE also uses a technique called overclustering, where the number of clusters $K_1$ is larger than the actual number of semantic categories.

Overclustering helps the network avoid collapsing multiple semantic categories into a single cluster. The final loss includes a balancing term:

$$L = \lambda_{k_1} L_{k1} + \lambda_{k_2} L_{k2} \tag{109}$$

Where $k_1$ and $k_2$ are the number of clusters, and $\lambda_{k_1}$ and $\lambda_{k_2}$ are balancing weights. The idea is to balance the clustering objective across different cluster granularities.

## 4.6 STEGO

The paper "Unsupervised Semantic Segmentation by Distilling Feature Correspondences" [16] proposes a method called STEGO (Self-supervised Transformer with Energy-based Graph Optimization). It is designed for unsupervised semantic segmentation. Given an image $I$, the algorithm extracts visual features using DINO such that $f = DINO(I) \in \mathbb{R}^{CxH}xW$. These features are mapped to a lightweight neural network S $\left(f \in \mathbb{R}^{KxH}xW\right)$ which map f into lowerdimensional embedding space where each feature corresponds to a particular semantic category. The embedding $S(f)$ is designed to cluster features that belong to the same semantic class together. For example, pixels corresponding to the cat's face, body, and ears will be mapped to a similar region in the embedding space, indicating they all belong to the "cat" category. After the segmentation head maps features into the new code space, the algorithm applies a clustering step (like K-Means) to assign discrete semantic labels to the compact feature clusters. These clusters represent the final semantic segments of the image

**Detailed Methodology:** STEGO uses a pretrained self-supervised model (such as DINO) to extract dense feature maps from input images. The feature maps for two images x and y are denoted as $f(x) \in \mathbb{R}^{C\times H} \times W$ and $f(y) \in \mathbb{R}^{CxH}xW$ where C represents the number of channels, and H and W represent the spatial dimensions. For two images $x$ and $y$, STEGO computes the feature correspondence tensor F:

$$F_{hwij} = \sum_c \frac{f_{chw}}{|f_{hw}|} \frac{g_{cij}}{|g_{ij}|}$$

Here, $f_{chw} \in \mathbb{R}^{CxHxW}$ is the feature vector at pixel location $(h, w)$ of image x and $g_{cij} \in \mathbb{R}^{CXIXJ}$ is the feature vector at pixel location $(i, j)$ of image y.C is the channel dimension, and the spatial dimensions are represented by $(h, w)$ and $(i, j)$. The equation computes the cosine similarity between features in different spatial locations. This tensor helps capture the similarity between features at different spatial locations and is used to define correspondences between images.

STEGO learns a lightweight segmentation head S that projects the features $f(x)$ and $f(y)$ into a lower-dimensional space $s \in \mathbb{R}^{kxhxw}$ where K is smaller than the number of channels in the original feature space. The segmentation head's goal is to refine the feature correspondences and ensure they form compact clusters.

Let $s(x) = S(f(x))$ and $t(y) = S(f(y))$ be the projection outputs for images $x$ and $y$ respectively. The similarity between projected features is used to compute the segmentation correspondence tensor $S$ :

$$S_{hwij} = \sum_c \frac{s_{chw}}{\|s_{chw}\|} \cdot \frac{t_{cij}}{\|t_{cij}\|} \tag{110}$$

STEGO's central loss function compares the feature correspondence tensor $F$ and the segmentation correspondence tensor $S$ for the same pair of images. The objective is to align the segmentation features with the pretrained features. The simple correlation loss is defined as:

$$L_{\text{simple-corr}}\left(x,y,b\right)=\sum_{hwij}\left(F_{hwij}-b\right)S_{hwij} \tag{111}$$

Where b is a bias term added to prevent collapse by introducing "negative pressure". The aim is to maximize the agreement between F and S where the feature correspondences are strong, and to push them apart where the feature correspondences are weak.

**Feature Centering & Clamping:** To stabilize the optimization, the paper introduces two modifications:'
Spatial Centering: Adjusts the correspondence tensor F by centering it spatially to handle small objects better:

$$F^{SC}_{hwij}=F_{hwij}=\frac{1}{IJ}\sum_{i'j'}F_{hwi'j'}$$

Zero Clamping: Ensures that segmentation correspondences do not encourage total antialignment by clamping segmentation correspondences to zero when they are weakly correlated:

$$L_{corr}(x,y,b)=-\sum_{hwij}\left(F^{SC}_{hwij}-b\right)\max\left(S_{hwij},0\right)$$

These two modifications improve the stability of the loss function and ensure that smaller objects are better handled.

The correspondence distillation loss can be seen as minimizing an energy function over a graph of image pixels, akin to the Potts model:

STEGO further refines the learned correspondences by introducing KNNs. For each image, the algorithm finds its K-nearest neighbors in the feature space and computes the loss for each neighbor.

## 4.7 FDA

The intuitive idea behind the Fourier Domain Adaptation (FDA) paper [34] is centered around a simple but effective way to bridge the gap between two domains (e.g., synthetic images and realworld images) by focusing on the low-level image features that don't impact the high-level semantic content.

The core intuition of the paper is that low-frequency information in an image (such as global lighting, color, or texture) is often responsible for these domain shifts. However, high-frequency information (like edges, shapes, and object boundaries) contains the essential information for understanding the image's semantic content (e.g., identifying a car, tree, or person).

So, the idea is:

- Low-frequency features (colors, lighting conditions) can be swapped between the source and target domains without affecting the high-level semantics (what objects are present in the image).

- By replacing the low-frequency content in the source image with the low-frequency content from the target image, we make the source image look more like a target image in appearance, while keeping the high-level content unchanged.

Fourier Transform: First, the source and target images are converted into the frequency domain using the Fourier transform. This breaks the images into low-frequency components (global features like color, brightness) and high-frequency components (fine details like edges and textures). The method swaps the low-frequency components of the source image with those from the target image, while keeping the high-frequency components (which contain the key semantic information) intact. This makes the source image "look" like a target image in terms of basic appearance (colors, lighting), but it keeps the important high-frequency information unchanged.

In the paper, the Fourier transform is applied to images to separate them into frequency components. For an image x of size $HxW$, the Fourier Transform is defined as:

$$F(x)(m,n) = \sum_{h,w} x(h,w) e^{-j2\pi\left(\frac{h}{H}m + \frac{w}{w}n\right)} \tag{112}$$

Here $F(x)(m,n)$ is the Fourier coefficient for the frequency pair and $e^{-j2\pi(,)}$ is the complex exponential. After the Fourier transform, the image is represented in terms of amplitude and phase components:

- Amplitude FA(x): The magnitude of the frequency component.
- Phase FP(x): The angle of the frequency component.

Mask $M_\beta$ to adjust the low-frequency components. This mask is defined around the lowfrequency region:

$$M_\beta(h,w) = 1_{(h,w)\in[\beta H;\beta H,-\beta W;\beta W]}$$

The parameter $\beta$ controls the size of the low-frequency region to swap. The core operation of FDA is to replace the low-frequency amplitude of the source image $x_s$ with that of the target image $x_t$ :

$$xs \to t = F^{-1}\left(\left[M_\beta \circ FA(xt) + (1 - M_\beta) \circ FA(xs), FP(xs)\right]\right) \tag{113}$$

Here, $FA(xt)$ is the amplitude of the target image, and $FP(xs)$ is the phase of the source image. The inverse Fourier transform $F^{-1}$ converts the modified spectrum back to the image domain, creating an image $xs \to t$ that retains the structure of the source image but has the appearance of the target image.

After performing FDA, the transformed source images $xs \to t$ are used to train a segmentation network $\phi_w$ using a standard-cross entropy loss:

$$L_{ce}\left(\phi_w; D_{s\to t}\right) = -\sum_i \left\langle y_s^i, \log\left(\phi_w\left(xs \to t^i\right)\right)\right\rangle \tag{114}$$

This loss function encourages the network to predict the correct segmentation labels for the transformed source images.

The model also incorporates an entropy minimization loss on the target images to regularize the decision boundary:

$$L_{ent}\left(\phi_w; D_t\right) = \sum_i \rho\left(-\left\langle \phi_w\left(xt^i\right), \log\left(\phi_w\left(xt^i\right)\right)\right\rangle\right) \tag{115}$$

Here, $\rho(\text{x}) = \left(x^2 + 0.001^2\right) \eta$ is a robust penalty function (Charbonnier loss) that penalizes high-entropy (uncertain) predictions.

## 4.8 Semantic Correspondence

**Introduction to Semantic Correspondence:** Semantic correspondence refers to establishing dense, pixel-level associations between semantically related regions in two or more images. Unlike traditional image alignment techniques, which rely on geometric transformations, semantic correspondence focuses on identifying regions that share similar semantic content even if they differ in appearance, scale, or pose.

**Representing the Task:** Given two input images $I_1$ and $I_2$ of sizes $W_1 \times H_1 \times C$ and $W_2 \times H_2 \times C$, respectively, the goal is to compute a correspondence map $C(x_1, y_1) \to (x_2, y_2)$ that aligns each pixel $(x_1, y_1)$ in $I_1$ to a corresponding pixel $(x_2, y_2)$ in $I_2$ based on semantic similarity. This mapping is expressed as:

$$C(x_1, y_1) = \underset{(x_2, y_2)}{\operatorname{argmax}} \, \phi(F_1(x_1, y_1), F_2(x_2, y_2)) \tag{116}$$

where $F_1$ and $F_2$ are feature maps extracted from $I_1$ and $I_2$, respectively, and $\phi$ is a similarity function that measures semantic compatibility between features.

To understand the framework, suppose we have two $3 \times 3$ matrices $A$ and $B$, representing two images. Each element in these matrices corresponds to the intensity of a pixel. Semantic correspondence maps pixels from $A$ to $B$ based on a similarity metric or structural context.

**Matrix $A$: Original Image**

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

**Matrix $B$: Transformed Image**

$$B = \begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix}$$

Matrix $B$ is a rotated version of $A$ (90° clockwise). Semantic correspondence is to find which pixel in $A$ corresponds to each pixel in $B$. For semantic correspondence, the mapping can be represented as:

From A (row, col) → To B (row, col)

$$(1,1) \to (1,3), \quad (1,2) \to (2,3), \quad (1,3) \to (3,3),$$

$$(2,1) \to (1,2), \quad (2,2) \to (2,2), \quad (2,3) \to (3,2),$$

$$(3,1) \to (1,1), \quad (3,2) \to (2,1), \quad (3,3) \to (3,1).$$

A simple example of correspondence is given two images of the same car taken from different angles, find the car's headlights in one image to the headlights in the other. Even though the positions or orientations of headlights differ, a correct algorithm should be able to align features based on meaning and not just appearance or location. Thus, the losses employed in correspondence tasks should ensure accurate matching of semantically similar regions while preserving structural and geometric properties across images.

Often, the typical losses model the shift from source to target image as a transformation. Given this, **correspondence loss** is defined as thus,

$$\mathcal{L}_{\text{correspondence}} = \frac{1}{N}\sum_{i=1}^{N} \|T_s(x_i) - x_i'\|_2^2 \tag{117}$$

Where $T_s$ is transformation applied to source points. Another natural choice is the **Cycle-Consistency Loss** which enforces that transformations applied forward and backward map points back to their original positions, ensuring consistency:

$$\mathcal{L}_{\text{cycle}} = \frac{1}{N}\sum_{i=1}^{N} \|T_t(T_s(x_i)) - x_i\|_2^2$$

Where $T_s$ is Source-to-target transformation and $T_t$ is Target-to-source transformation. This ensures the mappings are bijective and robust. Since correspondence requires a braoder understanding of meaning, **semantic alignment** loss often aligns high-dimensional feature embeddings of semantically similar regions between images:

$$\mathcal{L}_{\text{alignment}} = \frac{1}{N}\sum_{i=1}^{N} \|\phi_s(x_i) - \phi_t(x_i')\|_2^2 \tag{118}$$

Where $\phi_s$ is the Feature embedding function for the source image. and $\phi_t$ is the Feature embedding function for the target image.

This ensures semantically similar regions have consistent feature representations.

## 4.9 Unsupervised Semantic Correspondence Using Stable Diffusion

Let us consider the paper, "Unsupervised Semantic Correspondence Using Stable Diffusion" [17]. The main point of the paper is that pre-trained diffusion models specifically Stable Diffusion can be used for tasks beyond image generation-specifically for finding semantic correspondences between images, without any additional training or supervision.

Firstly, let us understand how we can define **local semantic correspondence using attention maps**. Let $\mathcal{J}_s$ and $\mathcal{J}_t$ represent the source image and target image, respectively. Let $p_s \in \mathcal{J}_s$ be the query point in the source image. We aim to find areas in the target image $\mathcal{J}_t$ that semantically correspond to $p_s$. First, we extract feature representations from both images. Assume we have a feature extractor $f\ (.)$ (e.g., a CNN or Transformer backbone) that maps each pixel or patch in the image to a high-dimensional feature space:

$$F_s = f\left(\mathcal{J}_S\right), F_t = f\left(\mathcal{J}_t\right) \tag{119}$$

Where $F_s$ and $F_t$ are feature maps corresponding to the source and target images. Each feature map represents the features at various spatial locations in the respective images.

Let $f\left(p_s\right) \in \mathbb{R}^d$ be the feature vector corresponding to the query point $p_s$ in the source image, where d is the dimension of the feature space. We now compute an attention map to highlight areas in the target image that correspond to the query point $p_s$. To compute this, we calculate a similarity score between the feature vector $f\left(p_s\right)$ and the feature vectors at every position in the target image feature map $F_t$. This could be done using a dot product or cosine similarity. For simplicity, we use the dot product here:

$$\operatorname{sim}\left(f_s\left(p_s\right), f_t\left(J_t\right)\right) \tag{120}$$

This similarity score measures how aligned or "similar" the feature representations of the query point in the source image and the points in the target image are. To create an attention map, we normalize the similarity scores using a softmax function over all positions in the target image:

The final attention map is a 2 D grid of attention weights $\alpha\left(p_t\right)$ which highlights the regions in the target image that are most semantically similar to the query $p_s$ in the source image. In essence:

$$A_t\left(p_t\right)=\left\{A_t\left(f_s\left(p_s\right), f_t\left(\mathcal{J}_t\right)\right) \mid p_t \in I_t\right\} \tag{121}$$

The authors in the paper use stable diffusion model II as their feature extractors $f(.)$. In particular, they utilize the cross-attention maps $A_{\text{cross}}$ . Given a source image $\mathcal{J}^S$ and a query point $u$ in the image $\mathcal{J}^S$ (such as a paw), they optimize an embedding $e$ so that the attention map $A_t$ of that embedding $e$ with the source image $\mathcal{J}^s$ highlights the query location in the source image.

During inference, this optimized embedding $e$ creates a cross-attention matrix map $A_\alpha$ with the target image $\mathcal{J}^t$. They take the maximum of this attention map $A_\alpha$ to find the location $u_p$ that points to a region semantically similar to $u$ (see the figure below)

**Method:** Given an image $I$, we use a VQ-VAE to map it to its encoder representation $\vec{z}_0(t)$. Subsequently, we noise it using the forward process $\vec{z}_T(t)$. We then perform denoising using the U-NeT to map $\vec{z}_t(t)$ to $\vec{z}_{t-1}(t)$ progressively until we obtain the latent $\vec{z}_0(t)$. This conditioning is performed using a text-embedding $\vec{e}$ which is randomly initialized. In particular, if you look at the code, it is defined inside "optimize_prompt" with the variable context $= \vec{e}$ being the vector that is optimized. The cross-attention at the $l-th$ layer of the U-Net within the diffusion model is defined as:

$$M_l(\vec{e}, l)=\operatorname{softmax}\left(\frac{Q_l K_l^T}{\sqrt{d_l}}\right) \tag{122}$$

Where, $Q_l=\phi_l(\vec{z}(t)) \in \mathbb{R}^{Cx(h \times w)xd_l}$ is the query obtained from the image features $\vec{z}(t)$, $K_l=\psi_l(\vec{e}) \in \mathbb{R}^{CxPxd_l}$ is the key obtained from the text embedding $\vec{e}$, $d_l$ is the dimension of the latent space at layer $l$ and $M_l(e, l) \in \mathbb{R}^{Cx(hxw)xP}$ is the attention map, with h, w representing spatial dimensions, P is the number of textual tokens and C the number of attention heads.

To reiterate, the attention map $M_l(e, l)$ assumes the following form $M_l(e, l) \in \mathbb{R}^{Cx(hxw)xP}$ where h x w are the spatial dimensions, P is the number of textual tokens and is the number of heads

The first thing that authors do is that they average across the Channels or number of heads. As a result, the attention heads become:

$$M_c(e, l)=\operatorname{avg}_{\text{channel}}\left(M_l(e, l)\right) \in \mathbb{R}^{(hxw)xP} \tag{123}$$

To take advantage of the different characteristics of each layer, the authors average along both the channel axis and across a subset of U -Net layers $M_c(e, l) \in \mathbb{R}^{(hxw)xP}$. Obviously, the size of attention maps is different, so what they do is they perform bilinear interpolation when averaging. Thus, they obtain:

$$M_{u-net} = avg_{\text{unet}}\ (M_c(e, l)) \in \mathbb{R}^{(hxw)xP} \tag{124}$$

In the code, the function responsible for performing bilinear interpolation is `upscale_to_image_size`. When handling an attention map comprising 1024 tokens, corresponding to a 32 × 32 grid, this function first reshapes the 1024 tokens into a 32 × 32 grid. Subsequently, it applies bilinear interpolation to upscale the 32 × 32 grid to a 512 × 512 resolution.

Finally, the authors pick an attention map associated with the first text token p. In particular,

$$M''\,(u_s, I, e) = M_{u-net}[1] \in \mathbb{R}^{(hxw)} \tag{125}$$

Where the authors do not employ the first or last token as typically these are special termination token. They also state that they empirically observed that the choice of which token does not have a significant impact on the final outcome as all other tokens (i.e., $P$ entries of $\vec{e}$ ) all are also optimized regardless due to the softmax that we apply along the P axis - optimization will find the prompts (or more exactly the embeddings) that match the chosen token location.

**Optimization:** For a given source image $I_s$ and a query location $u_s \in [0, 1]^2$, the authors aim to optimize a text embedding e such that the attention map $M\,(u; e, I_s)$ focuses on the region of interested centered around $u_s$. To understand this, imagine you have two images:

1. Source image $I_i$ : An image of a cat.
2. Target Image $I_j$ : An image of a dog.

We want to find a pixel in the dog image $\boldsymbol{I}_j$ that corresponds to a specific pixel (say, the pixel that represents the cat's ear) in the cat image $\boldsymbol{I}_i$. The task is to find the pixel in $\boldsymbol{I}_j$ that represents the dog's ear, which is semantically similar to the cat's ear in the source image.

Let us say we have a query pixel $u_i$ in the cat image that corresponds to the cat's ear. The pixel location is normalized to the image size, meaning it points to a specific pixel in the source image (e.g., normalized coordinates might be $\boldsymbol{u}_i = (0.3, 0.5)$ which corresponds to the center of the cat's ear). The task is to find the semantically corresponding pixel $u_j$ in the dog image $I_j$ which should point to the dog's ear.

Now, we already have $M''\,(u_s, I, e)$. The problem with this map is that it might be too spread out or not focused on the specific region we care about because $e$ was initialized randomly. The model's attention could be distributed across many irrelevant areas in the image. For example, if we are trying to focus on the dog's ear, the initial attention map might focus on multiple regions (like the dog's tail, legs, etc.), because the model hasn't yet learned to concentrate its attention on the specific query pixel (remember, we used a random query e for now).

We must now learn to make the attention map of $e$ which is $M''\,(u_s, I, e)$ to focus on the region of our interest. In order to do this, we define our region of interest $u_s$ in the source image. At this query location $u_s$, we define a Gaussian of standard deviation $\sigma$ centered at $u_s$

$$M_S(u) = \exp\left(-\frac{\|u - u_S\|_2^2}{2\sigma^2}\right) \tag{126}$$

$M_s(u)$ is calculated for every pixel u in the image, representing the attention assigned to each pixel based on its distance from the query pixel $u_i$. Thus, given that the query pixel is dog's ear, centered at $(100, 150)$, we would define $M_s(u)$ over the entire ( $512 \times 512$ grid). The optimization objective is to find the text embedding $\vec{e}^{*}$ that minimizes the difference between the model's unfocused attention map $M''\left(u_s, I, e\right)$ and the Gaussian map that focuses on our region of interest $M_S(u)$ :

$$\vec{e}^{*} = \operatorname{argmin}_{\vec{e}} \sum_u \left\|M''\left(u; e, I_s\right) - M_s(u)\right\|_2^2 \tag{127}$$

Assume the dog's ear is located at pixel $u_s = (100, 150)$ in the image. We want the model to focus its attention primarily on this pixel and nearby pixels. To do this, we define a Gaussian distribution centered at the pixel $u_i$ : $M_i(u) = \exp\left(-\frac{\|u-u_i\|_2^2}{2\sigma^2}\right)$. Here, $u$ represents all the pixel locations in the image (for instance, all possible u = (x, y) pixel pairs). The Gaussian map $M_i(u)$
will have the highest value at the dog's ear and gradually taper off as you move away from the ear. If we set $\sigma = 10$, the Gaussian distribution will have a tight focus around the dog's ear.

The optimization process in equation (6) adjusts the text embedding $e$ to minimize the difference between the attention map $M''\left(u; e, I_s\right)$ and $M_s(u)$.

Is the MSE loss enough to capture the fine-grained patterns of the attention map at the query $q$ ?

Once the optimized text embedding $\vec{e}^{*}$ is obtained, it can be applied to a target image $I_t$ to find the semantically corresponding region. The corresponding pixel $u_t$ in the target image is determined by finding the maximum attention value:

$$u_t = \operatorname{argmax}_u M\left(u; e^*, I_t\right) \tag{128}$$

This step involves computing the attention map for the target image and identifying the pixel with the highest attention score, which corresponds to the region that is semantically similar to the query region in the source image.

**Optimization across transformations:** The optimization of text embeddings on a single image is prone to overfitting. To address this issue, the authors propose averaging across image crops. For example, when optimizing the attention map for a specific region, such as the dog's ear in the source image, the training process involves presenting cropped versions of the source image, where only 93% of the image is visible. For each crop, the model generates an attention map, repositions it back into the full image's coordinates, and compares it to a Gaussian map that is similarly cropped and repositioned. The cropping is applied randomly across the image, ensuring that other regions maintain their relative positions.

The model adjusts its parameters to align the attention map with the Gaussian map across various crops, allowing it to focus on the target region, such as the dog's ear, regardless of cropping. During inference, a similar averaging strategy is employed to predict the corresponding pixel $u_j$ in the target image $I_t$. For a given crop $\mathcal{C}$, the operation $\mathcal{U}$ places the attention map for the cropped image back into the full image's coordinate system. Attention maps are generated for multiple cropped views of the target image, repositioned, and averaged to produce a robust attention map. The final step involves taking the argmax over the pixel locations $u$ to determine $u_j$, the pixel in the target image with the highest attention, corresponding to the semantic match for the query pixel in the source image.

**Averaging Across Crops** Let $\mathcal{C}(I)$ represent the cropping operation with parameters $c$, and $\mathcal{U}_c(x_c)$ denote the placement of the crop back to its original location, satisfying $\mathcal{C}_c(\mathcal{U}_c(x_c)) = x_c$ for some crop $x_c$. The image dimensions are reduced to 93% (determined via hyperparameter tuning), and a uniformly random translation is applied, denoted as $c \sim D$. The optimization process in Equation (6) is augmented by averaging across cropping augmentations as:

$$\vec{e}^{*} = \text{argmax}_{\vec{e}}\, \mathbb{E}_{c\sim D} \sum_{u} \left\| \mathcal{C}_c\left(M''(u; e, I_s)\right) - \mathcal{C}_c\left(M_s(u)\right) \right\|_2^2 \tag{129}$$

Similarly, during inference, attention masks from different crops are averaged:

$$u_j = \text{argmax}_u\, \mathbb{E}_{c\sim D} u_c\left(M(u; e^{*}, I_t)\right) \tag{130}$$

**Averaging Across Optimization Rounds** Empirical results show that performing multiple rounds of optimization is crucial for achieving strong performance, as illustrated in the figure below.

The optimization process in Equation (8) can be abstracted as $\vec{e}^{*} = \mathcal{O}(\bar{e}, I_i)$, where $\bar{e}$ represents the initialization. The attention masks obtained from multiple optimization runs are then averaged to improve robustness:

$$u_j = \text{argmax}_u\, \mathbb{E}_{e\sim D} M(u; \mathcal{O}(\bar{e}, I_i), I_J) \tag{131}$$

Each optimization round begins with a different initialization of the embedding $\vec{e}_j$, and the results are aggregated to produce a more reliable attention map. This approach ensures that the attention maps are robust to variations in initialization and cropping, resulting in better alignment with the target regions.

### 4.10 Unsupervised Keypoints

The paper, "Unsupervised Keypoints from Pretrained Diffusion Models," [17] build on the previous work of utilizing the already-encoded semantic knowledge encoded in Diffusion Models for downstream tasks. The main objective is to locate "keypoints" in images—distinct and semantically important regions—without relying on annotated data.

These Keypoints are essential in computer vision tasks like 3D reconstruction and motion tracking. These points serve as essential features that represent the geometry and structure of objects. One of their key advantages is transformation invariance, meaning they remain stable under scaling, rotation, and translation. This stability allows keypoints to generalize well, helping algorithms recognize objects regardless of context or perspective.

Keypoints can be classified into two types: supervised and unsupervised. Supervised keypoints are determined using annotated data where humans specify important landmarks, such as the corners of the mouth or the position of the eyes in facial recognition tasks. In contrast, unsupervised keypoints are automatically discovered without labeled data, relying on models to learn patterns or properties directly from images.

An example of keypoints' application is in human pose estimation, such as tracking a person in a video for activity recognition (e.g., Tai Chi movements). In a supervised approach, the method would rely on annotated datasets where humans have labeled joints like the head, shoulders, elbows, and knees in thousands of frames. These labeled landmarks serve as the ground truth for training. On the other hand, the unsupervised approach discussed in this paper avoids the need for annotations. Instead, the model identifies consistent patterns across multiple images of people performing Tai Chi.

The method relies on the cross-attention layers within diffusion models, where each text embedding token $\vec{e}$ connects to specific areas in the image, captured in an attention map $M(\vec{e}, X)$. The attention map is calculated as:

$$M(\vec{e}, \mathrm{X}) = E_C \left[ \mathrm{softmax}_n \left( \frac{Q_L^c \cdot K_l^c}{\sqrt{D_l}} \right) \right] \tag{132}$$

Where $E_c$ denotes averaging over transformer heads c.

The model begins with attention maps $M(\vec{e}, X)$ that are dispersed and exhibit activity spread across the entire image. To ensure the attention maps focus on specific, compact regions, the method introduces a localization mechanism. This mechanism works by defining a Gaussian distribution $G_n$ centered at the peak location of the attention map $M_n$ for each token $n$.

The process starts by identifying the peak location of $M_n$, which corresponds to the pixel with the highest response in the attention map. This peak, $\mu_n$, is mathematically calculated as:

$$\mu_n = \mathrm{argmax}_{w,h} \, M_n[h, w] \tag{133}$$

Using this peak, a Gaussian $G_n$ is created, centered at $\mu_n$, and is expressed as:

$$G_n = \exp\left( -\frac{\|XY_{\mathrm{coord}} - \mu_n\|^2}{2\sigma^2} \right) \tag{134}$$

Here, $XY_{\mathrm{coord}}$ represents the spatial coordinates of the image and $\sigma$ controls the spread of the Gaussian distribution. The Gaussian $G_n$ serves as the ideal localized representation that the attention map $M_n$ should emulate.

To encourage this localization behavior, a localization loss $\mathcal{L}_{\mathrm{localize}}$ is defined, which measures the difference between the attention map $M_n$ and the target Gaussian $G_n$. This loss is calculated as:

$$\mathcal{L}_{\mathrm{localize}} = \mathbb{E}_n \|M_n - G_n\|^2 \tag{135}$$

This loss compels the attention maps to become compact and focused, aligning closely with the Gaussian distributions centered at their respective peaks. In addition to localization, the method enforces consistency of attention maps under image transformations. This is achieved through the equivariance loss $\mathcal{L}_{\mathrm{equiv}}$, which ensures that the attention maps remain consistent when the input images undergo small transformations $T$, such as rotations, translations, and scaling. The equivariance loss is expressed as:

$$\mathcal{L}_{\mathrm{equiv}} = \mathbb{E}_n \|T^{-1}(M_e(e, T(X))) - M_n(e, X)\|^2 \tag{136}$$

This loss ensures that the model maintains a stable and reliable representation of keypoints, regardless of minor geometric changes in the input images.

The overall training objective combines these two losses into a single objective function:

$$L_{\mathrm{total}} = \mathcal{L}_{\mathrm{localize}} + \lambda_{\mathrm{equiv}} \mathcal{L}_{\mathrm{equiv}} \tag{137}$$

Here, $\lambda_{\mathrm{equiv}}$ is a hyperparameter that balances the contribution of the equivariance loss relative to the localization loss. This combined objective ensures that the attention maps are both localized and robust to transformations, enabling the model to identify semantically meaningful and consistent keypoints across diverse images. This methodology allows the model to generalize effectively, even when the input data varies significantly in pose, scale, or background conditions.

## 4.11 Self-Attention Guidance

**Self-attention maps** play a critical role in semantic segmentation within diffusion models by enabling fine-grained feature representation. These maps capture the pairwise relationships between every pixel in an image, allowing the model to learn how each region of the image influences others. This is particularly essential in semantic segmentation where understanding the global context and dependencies between spatial regions is necessary for accurate delineation of object boundaries and classes. Keeping this in mind, the work, "Improving Sample Quality of Diffusion Models Using Self-Attention Guidance" [19] is of special interest since it utilizes self-attention maps for downstream tasks.

Recall that there are two types of guidance: **Classifier Guidance (CG)** and **Classifier-Free Guidance (CFG)**. CG uses a trained classifier $p(c|\mathbf{x}_t)$ to guide the reverse process toward specific class distributions. The noise prediction is modified as follows:

$$\tilde{\epsilon}(\mathbf{x}_t, c) = \epsilon_\theta(\mathbf{x}_t) - s\sigma_t \nabla_{\mathbf{x}_t} \log p(c|\mathbf{x}_t),$$

where $s$ is the guidance scale controlling the strength of the classifier's influence.

CFG simplifies the guidance process by interpolating between conditional and unconditional predictions, formulated as:

$$\tilde{\epsilon}(\mathbf{x}_t, c) = \epsilon_\theta(\mathbf{x}_t) + (1+s)(\epsilon_\theta(\mathbf{x}_t, c) - \epsilon_\theta(\mathbf{x}_t)).$$

CFG does not require a separate classifier but relies on external labels or prompts, making it inapplicable to fully unconditional models. The proposed Self-Attention Guidance (SAG) method utilizes internal self-attention maps to guide the diffusion process. Unlike CG or CFG, SAG is condition-free and relies on internal representations. The guidance noise is computed as:

$$\tilde{\epsilon}(\mathbf{x}_t) = \epsilon_\theta(\mathbf{x}_t) + (1+s)(\epsilon_\theta(\mathbf{x}_t) - \epsilon_\theta(\widehat{\mathbf{x}}_t)), \tag{138}$$

where $\widehat{\mathbf{x}}_t$ is the selectively blurred version of $\mathbf{x}_t$, determined by the self-attention map $A_t$.

The aggregated self-attention map is obtained via global average pooling (GAP):

$$A_t = \text{Upsample}(\text{Reshape}(\text{GAP}(A_t^S))), \tag{139}$$

where $A_t^S$ denotes the stacked self-attention maps across heads.

To understand how selective self-guidance is employed in 138, consider the self-attention map $A_t$ at each timestep t . The attention map encodes information about which regions of the image or feature map are most important for the model's current prediction. Once the attention map $A_t$ is computed, the next step is to decide which regions are important enough to be blurred. This is done by applying a threshold $\psi$ to the attention map. Specifically, the regions that have attention scores above the threshold are considered "important" and are the ones that will be blurred. Mathematically, this is represented as:

$$M_t = 1\,(A_t > \psi) \tag{140}$$

Where $M_t$ is a binary mask created from the attention map, and 1 is the indicator function that assigns a value of 1 to regions where $A_t > \psi$ and 0 otherwise. After creating the mask $M_t$ the regions identified by the mask where $M_t = 1$ are blurred. The idea here is to blur only the regions that the model attends to the most-those areas that the model considers important for image generation. By doing this, the model is forced to refine these regions further during the denoising process.

The blurred version $\tilde{x}_t$ of the image is combined with the original image $x_t$ using the mask $M_t$ :

$$\tilde{x}_t = (1 - M_t) \odot x_t + M_t \odot \tilde{x}_t \tag{141}$$

Blurring the most important regions (as identified by the attention map) creates an adversarial scenario. The model is effectively challenged to restore the fine details in the blurred regions during the denoising process. Since these regions are crucial for image generation, the model is forced to focus more on refining and improving them, leading to higher-quality output. This is an important aspect of SAG: it forces the model to pay more attention to important areas by intentionally blurring them and then requiring the model to restore these regions during the diffusion process.

Empirical results show that SAG improves sample quality across various models which include ADM and Stable Diffusion. Metrics such as FID and IS indicate consistent improvements. Moreover, SAG is shown to be orthogonal to CG and CFG and complements them.

## 4.12 Label-Efficient Segmentation

The paper "Label-Efficient Semantic Segmentation with Diffusion Models" [5] shows how Diffusion Probabilistic models (DDPMs) can extract meaningful pixel-level representations for image segmentation in low-data settings.

In this paper, **representation learning** is applied by leveraging the intermediate activations of DDPMs during the reverse diffusion process as pixel-wise semantic features for segmentation tasks. Thus, it shows that these features are *generalizable* to the task of segmentation even though DDPMs were originally trained for generative purposes.

For a noised latent $\vec{z}_T$, the diffusion model's noise prediction network (often parameterized by a U-Net) produces a series of feature maps $f_1, f_2, ..., f_n$ across the different layers of the U-Net architecture at each timestep t . Let $f_\theta\left(\vec{z}_t, t\right)$ be the feature map produced by the U-Net decoder at timestep $t$ and $B_i$ denote the feature map from the i-th block of the U-Net decoder. The feature map at block $B_i$ will have a spatial resolution $H_i \times W_i$ which is typically smaller than the input resolution $HxW$. The feature maps are upsampled to the original image resolution $HxW$ using bilinear interpolation to form pixel-level representations:

$$\hat{f}_\theta\left(\vec{z}_i, B_i\right) \in \mathbb{R}^{H \times W \times C_i} \tag{142}$$

Where $C_i$ is the number of channels in the feature map. For segmentation, the authors extract representations from the middle layers of the U-Net decoder across several reverse diffusion steps:

$$\left\{B_5, B_6, B_7, B_8, B_{12}\right\}, t \in \{50, 150, 250\} \tag{143}$$

These timesteps are chosen together with the middle-blocks of the U-NeT for features because empirically the authors find that they tend to capture the most useful semantic information for segmentation.

The feature maps from these layers are concatenated to form a high-dimensional pixel-wise feature vector $\vec{f_p} \in \mathbb{R}^{HxWxD}$ where $D = \sum_i C_i$ is the total number of channels

Once the pixel-level representations are extracted, the goal is to predict a semantic label for each pixel. Since the method operates in a few-shot learning regime, only a small number of labeled images are available. For each pixel $p$, the concatenated feature vector $\vec{f_p}$ is passed through a multi-layer perceptron (MLP) classifier, which predicts the pixel's class label:

$$\hat{y}_i = MLP\left(\vec{f_p}\right) \tag{144}$$

Where $\vec{y_p} = \{1, 2, \ldots K\}$ represents the predicted class label for pixel p and K is the total number of semantic classes. The MLP consists of two hidden layers with ReLU activations and batch normalization, followed by a softmax output layer to predict the class probabilities.

The MLP classifiers are trained using the pixel-wise representations extracted from the few labeled images. The loss function used to train the MLP is the cross-entropy loss between the predicted and ground-truth pixel labels

$$\mathcal{L}_{\text{seg}} = \sum_{i=1}^{N} \text{CE}(y_i, \hat{y}_i) \tag{145}$$

where $y_i$ is the ground truth, $\hat{y}_i$ is the predicted label, and CE is the cross-entropy loss.

For a test image, we pass it through the DDPM and extract pixel-wise features. We then use the trained MLP to predict a class for each pixel. Using an ensemble of MLPs, we predict the final label using majority voting.

## 4.13 Diff-Attend Segment

**Non-Maximum Suppression:** Assume we have a set of $N$ candidate regions $\{R_1, R_2, \ldots, R_N\}$. Each region $R_i$ has:

1. **A confidence score** $S_i \in [0, 1]$ which measures how likely it is to contain the target object.
2. **Bounding box coordinates** $B_i$ often denoted as $(x_{\min}, y_{\min}, x_{\max}, y_{\max})$ for each box $i$.

The first step in NMS is to sort the regions by their confidence scores in descending order:

$$\{R_1, R_2, \ldots, R_n\} \quad \text{s.t.} \quad S_1 \geq S_2 \geq \cdots \geq S_N.$$

The Intersection over Union (IoU) measures the overlap between two bounding boxes, defined as:

$$\text{IoU}\left(B_i, B_j\right) = \frac{\text{Area}\left(B_i \cap B_j\right)}{\text{Area}\left(B_i \cup B_j\right)}.$$

Using an IoU threshold $T \in [0, 1]$, we proceed as follows:

1. Initialize an empty set $M$ to store the selected regions.
2. Iterate through each region in descending order of confidence score:
   (a) Select the current highest scoring region $R_i$ and add it to $M$.
   (b) For each remaining region $R_j$, calculate the IoU between $R_i$ and $R_j$:
   $$\text{IoU}\left(B_i, B_j\right).$$
3. Suppress $R_j$ if:
   $$\text{IoU}\left(B_i, B_j\right) > T.$$

Continue this process until all regions have either been selected or suppressed. At the end, $M$ contains the non-suppressed, highest-confidence regions.

**Example:** Let's do a simple example of NMS with 3 hypothetical regions $R_1, R_2$, and $R_3$, with confidence scores $S_1 = 0.9$, $S_2 = 0.8$, and $S_3 = 0.75$. Suppose the IoU threshold $T = 0.5$. Sort the regions by confidence scores: $S_1 = 0.9$, $S_2 = 0.8$, and $S_3 = 0.75$.
Start with $R_1$ (the region with the highest confidence score) and compare it with $R_2$ and $R_3$ using the IoU formula. Assume:

$$\text{IoU}(B_1, B_2) = 0.6 \quad \text{and} \quad \text{IoU}(B_1, B_3) = 0.4.$$

Since $\text{IoU}(B_1, B_2) > T$, suppress $R_2$. Since $\text{IoU}(B_1, B_3) < T$, keep $R_3$. Move to the next non-suppressed region, $R_3$, and repeat the process. After all comparisons, the final selected regions are:

$$M = \{R_1, R_3\}.$$

The paper, titled ”Diffuse, Attend, and Segment: Unsupervised Zero-Shot Segmentation using Stable Diffusion,” [29] introduces DiffSeg, an innovative method for unsupervised, zero-shot image segmentation. This approach leverages the self-attention layers of stable diffusion models to generate high-quality segmentation masks. The primary objective of DiffSeg is to enable the segmentation of any image without the need for pre-existing labels or supervision, utilizing the inherent ability of the self-attention layers to capture semantic groupings within their attention tensor

In the U-Net, each self-attention layer produces 4-dimensional tensors, which contain the spatial attention map for different resolutions. These attention tensors are denoted as:

$$\mathrm{A} \in \left\{A_k \in \mathbb{R}^{h_k \times w_k \times h_k \times w_k} \mid k = 1, \ldots, 16\right\} \tag{146}$$

where each tensor $A_k$ represents attention values at a particular resolution, with each dimension corresponding to spatial locations within that layer.

The authors introduce two properties of the attention maps which are essential for segmentation:

- **Intra-Attention Similarity**: Within a single 2D attention map $A_k[i, j, :, :]$ (for any given pixel (i,j)), pixels that belong to the same object tend to have high similarity in their attention values.
- **Inter-Attention Similarity**: Between different 2D attention maps, such as $A_k[i, j, :, :]$ and $A_k[i + 1, j + 1, :, :]$ similar attention values indicate that these areas likely belong to the same object across locations.

These properties suggest that self-attention layers inherently group image regions that share similar features, making them ideal candidates for segmentation. Each attention map, $A_k$, represents a different spatial resolution. To create a unified high-resolution representation, the attention maps are aggregated:

1. The attention maps are upsampled to the highest resolution, $64 \times 64$, using bilinear interpolation:

$$\tilde{A}_k = \text{ Bilinear-upsample } (A_k) \in \mathbb{R}^{h_k} \times w_k \times 64 \times 64 \tag{147}$$

2. Each upsampled attention map $\tilde{A}_k$ is assigned a weight $R_k$ proportional to its resolution, emphasizing high-resolution maps for detailed segmentation.

The final aggregated tensor, $A_f$, is calculated by summing these weighted attention maps:

$$A_f[i, j, :, :] = \sum_{k=1}^{16} \tilde{A}_k \left[\frac{i}{\delta_k}, \frac{j}{\delta_k}, :, :\right] \cdot R_k \tag{148}$$

Where $\delta = \frac{64}{w_k}$ controls the scaling for each resolution level, ensuring all attention maps are spatially aligned.

**Iterative Attention Merging:** This step leverages KL divergence to identify and merge regions of high similarity across the attention map:

1. Anchor Points: A grid of anchor points, M × M, is generated from the aggregated attention tensor $A_f$. Each anchor point has an associated attention map $\mathcal{L}_a$, serving as a segmentation ”seed”: $\mathcal{L}_a = \left\{A_f\left[i_m, j_m, :, :\right] \in \mathbb{R}^{64 \times 64} \mid (i_m, j_m) \in M\right\}$

2. Merging Criterion: The similarity between any two attention maps is measured using a symmetric KL divergence:

$$D\left(A_f[i,j], A_f[y,z]\right) = KL\left(A_f[i,j] \| A_f[y,z]\right) + KL\left(A_f[y,z] \| A_f[i,j]\right) \tag{149}$$

3. Iterative Merging: In each iteration, pairs of attention maps with similarity $D < \tau$ (a predefined threshold) are merged by averaging them. This iterative process gradually combines similar regions into unified object proposals, creating a list of object masks, $\mathcal{L}_p$

**Non-Maximum Suppression (NMS):** After merging, the remaining maps are refined to create the final segmentation mask. Each location in the image is assigned to the attention map with the highest probability, ensuring only the most likely segmentation label remains for each pixel. The resulting segmentation mask, $S \in \mathbb{R}^{64\times 64}$ is produced by subsampling $\mathcal{L}_p$ and then applying NMS:

$$S[i,j] = \arg\max \mathcal{L}_p[:, i, j] \tag{150}$$

To understand the framework, consider an example where we have an input image and we want to segment two main objects, say a cat and a dog. The image is processed by the U-Net component in Stable Diffusion. The U-Net generates 16 self-attention maps $A_k$ ( for k $= 1, 2, \ldots 16$ ) each representing a different spatial resolution. Attention maps come in four resolutions ( 8 x 8 ), ( $16 \times 16$ ), ( $32\times$ 32) and (64 x 64). DiffSeg aggregates these self-attention maps into a single high-resolution map $64 \times 64$ to combine information from all layers: $\tilde{A}_k =$ Bilinear Upsample $(A_k)$. We then assign higher weights to maps with finer details (higher resolution maps), and combine them into a single high-resolution attention tensor: $A_f[i, j, : \ . \ :] = \sum_{k=1}^{16} \tilde{A}_k[i, j, :, :] \times R_k$. Here $R_k$ is a weight that reflects the importance of each map. This results in an aggregated attention tensor $A_f$ of size $64 \times 64$. To make $A_f$ a probability distribution, normalize it such that each slice $A_f[i, j, : \ . \ :]$ sums to 1 . We then divide $A_f$ into anchor points by sampling locations on a grid. Let's say we create a $16 \times 16$ gride of anchors, so each anchor corresponds to a region within $A_f$. For each region $A_f[i, j, : \ . \ :]$ calculate the KL divergence with neighboring anchors. This measures how similar two attention maps are, indicating whether they belong to the same object. Start with the highest-scoring region and merge it with others that have a KL divergence below a threshold $\tau$. Continue merging until no more regions with a similarity below $\tau$ remains. This process iterates N times, gradually reducing the number of regions and forming coherent object groups. After iterative merging, we have a set of object proposals in the form of attention maps. NMS is used to assign each pixel to only one of these proposals. Each object proposal $L_p$ is unsampled to match the original image resolution. For each pixel $(i, j)$ look at the attention values across all object proposals $L_p[k, i, j]$. Assign the pixel to the proposal with the highest confidence:

$$S[i,j] = \arg\max_k L_p[k, i, j] \tag{151}$$

This ensures that each pixel belongs to only one object in the final segmentation map S. After applying NMS, S is a clean segmentation mask where each pixel is assigned to the most confident proposal, creating clear boundaries around each object (e.g., cat and dog).

## 4.14 Text Embeddings in T2I models

The embedding space of text in diffusion models is structured with specific dimensions to facilitate efficient processing. The size of an individual embedding vector is $[1, 768]$. However, diffusion models operate with a fixed number of tokens in each sequence, resulting in a sequence size of $[77, 768]$. This fixed token length is critical because, during text-conditioning, the attention mechanism is applied, which requires computing cross-attention from the key vector. Allowing the embedding sequence length to vary would mean that each text prompt could have a different number of tokens, resulting in variable shapes for each example in the batch. Such variability would complicate parallel processing and hinder GPU acceleration. To address this, even when the input consists of a single word, such as "dog," with an embedding dimension of $[1, 768]$, the sequence is padded to maintain the fixed length of $[77, 768]$, with the remaining tokens zero-padded.

**Contextual Correlations within Text Embeddings:** Insights from the paper *"Uncovering the Text Embedding in Text-to-Image Diffusion Models"* [35] highlight the contextual correlations within text embeddings in diffusion models. These correlations arise from how each word in a sequence influences others, governed by mechanisms such as the causal mask and the omission of the padding mask. These mechanisms play distinct roles in shaping the flow of context through the token sequence, thereby affecting the representation of both content (semantic embeddings) and style (padding embeddings).

**Causal Mask:** The causal mask enforces an autoregressive dependency within the sequence of token embeddings. In this setup, each token $e_i$ can only attend to the tokens that precede it in the sequence, such as $e_1, e_2, \ldots, e_{i-1}$, but cannot attend to tokens that follow. The causal mask modifies the attention matrix $A$ by setting the attention from a token $e_i$ to any future token $e_j$ (where $j > i$) to zero. Mathematically, this is represented as:

$$A_{i,j} = \begin{cases} \text{softmax}\left(\frac{Q_i K_j^T}{\sqrt{d}}\right) & \text{if } j \leq i \\ 0 & \text{if } j > i \end{cases} \tag{152}$$

This causal masking results in a context-ordered embedding, where each word embedding $e_i$ is influenced only by preceding words in the sequence. This approach effectively encodes a forward progression of information, with earlier words establishing broader context and subsequent words refining details. In diffusion models, such hierarchical encoding is particularly advantageous as it helps encode both global context and local details within a prompt.

**Padding Mask:** In standard transformer architectures, a padding mask is employed to ignore padding tokens—additional tokens added to shorter sequences to align them with the maximum sequence length within a batch. However, in diffusion models, the padding mask is omitted. This omission means that padding tokens are included in attention calculations and can interact with semantic tokens (tokens containing meaningful content). Consequently, padding tokens gain information from their neighboring semantic tokens. Mathematically, this interaction is expressed as:

$$A_{\text{pad},i} \neq 0 \tag{153}$$

This interaction enables padding tokens to acquire information that is not semantically significant but is instead related to style or background information.

**Token Representation in Diffusion Models:** The tokens generated by the diffusion model encoder can be categorized into two distinct types:

- **Semantic Embeddings:** These tokens carry meaningful content from the prompt, such as objects, actions, or descriptors, and primarily influence the content of the generated image.
- **Padding Embeddings:** These tokens, devoid of inherent semantic meaning, absorb stylistic or background information through their proximity to semantic tokens. Their influence is primarily on the style or overall appearance of the generated image.

This separation of token roles underscores how diffusion models effectively leverage text embeddings to balance content representation and stylistic refinement in image generation.

## 4.15 DiffewS

DiffewS [36] performs **few-shot image segmentation**. Imagine a scenario where a wildlife researcher has only a few annotated images of a rare species of bird, and they want to automatically segment images of this bird in a large collection of unlabeled photos. In this case, the Few-shot Semantic Segmentation (FSS) model in the paper would take a few labeled examples (support images of the bird with a mask highlighting the bird) and use them to guide segmentation on new, unlabeled images (query images).

DiffewS establishes a relationship between the bird's features in the support and query images. It modifies the attention mechanism to let the model focus on similar features across both images, helping it find the bird in new contexts or backgrounds. The support images have labeled masks showing the bird which helps the model understand what parts of the image are important. For instance it can focus on the bird's color, shape, and texture in the support images then apply this knowledge to detect similar regions in the query images.

The core of DiffewS is the **KV Fusion Self-Attention** which combines the key-value pairs from both support and query images in the UNet's self-attention mechanism. This fusion enables the model to focus on regions in the query image that are similar to the support image's annotated object.

For a given layer $l$ in the UNet, let:
$X_s^l$ : Feature map of the support image at layer $l$
$X_q^l$ : Feature map of the query image at layer $l$
The self-attention layer computes:

$$X_q^{l+1} = \mathrm{Attn} \cdot (Q_q, K_{qs}, V_{qs}) \tag{154}$$

Where $Q_q = X_q^l W_q$ is the query projection for the query image, $K_{qs} = [K_q, K_s] = \left[X_q^l W_k, X_s^l W_k\right]$ are the concatenated keys from query and support images and $V_{qs} = [V_q, V_s] = \left[X_q^l W_v, X_s^l W_v\right]$ are concatenated values from query and support images.

Thus, the attention mechanism is formulated as:

$$X_q^{l+1} = \mathrm{softmax}\left(\frac{Q_q K_{qs}^T}{\sqrt{d}}\right) V_{qs} \tag{155}$$

Let $I_s =$ Support image, $I_q =$ Query image, $M_s =$ The support mask, $M_q =$ The query mask. This is encoded in VQ-VAE to $z_s, z_q, z_{m_s}$ and $z_{m_q}$ respectively. To incorporate support mask information, the support mask $M_s$ is encoded into the latent space $z_s$ and combined with the support image's latent features. The paper evaluates several methods of integrating this mask information:

- Concatenation: $z_s$ and $z_{m_s}$ are concatenated in the channel dimension.
- Multiplication: Mask information is integrated by element-wise multiplication with the support latent.

The chosen approach influences how information about the object of interest in the support image is encoded, aiding the model in distinguishing relevant features for segmentation. During training, DiffewS optimizes for the accurate prediction of the segmentation mask for the query image. The model uses a modified objective function to train the UNet to produce a mask that closely matches the query mask $M_q$ :

$$L_{fss} = \mathbb{E}_{z_s, z_q, z_{m_s}, z_{m_q}} \left\| z_{m_q} - v_\theta \left( z_s, z_q, z_{m_s} \right) \right\|_2^2 \tag{156}$$

Where $v_\theta$ is the modified UNet model that receives the concatenated features from the support and query images (and masks) and outputs the segmentation mask latent for the query image.

**Tokenized Interaction Cross-Attention:** The Tokenized Interaction Cross-Attention approach in DiffewS leverages cross-attention to introduce information from the support image to the query image. Let the query image be the one which we want to segment. The support image, which has an annotated mask of the object of interest, provides the key-value features

To use the support image in cross-attention, it is first encoded into a set of tokens. The support image $I_s$ is passed through a CLIP image encoder, denoted as CLIP $_{\text{img}}$ . The CLIP encoder converts the image into a sequence of tokens (feature vectors):

$$\text{Token}_s = CLIP_{img} \left( I_s \right) \tag{157}$$

Where Token $_s \in \mathbb{R}^{Lxd}$ with $L$ being the number of tokens (spatial features) and d the dimension of each token. After encoding the support image, the resulting token sequence Token $_s$ is flattened to match the expected input for the cross-attention layer. Flattening here means that
the 2 D spatial structure of the tokens is reshaped into a single sequence, making it suitable for direct use in cross-attention.

This step is represented as:

$$\text{Flatten } \left( \text{ Tokens }_s \right) = \left[ t_1, t_2, \ldots t_L \right] \in \mathbb{R}^{Lxd}$$

Where $t_i$ are the individual tokens of the support image. The cross-attention mechanism is used to link the query image with the tokenized support image. Here's how the cross-attention operation is set up:

The query features $X_q$ from the query image are projected to create queries Q :

$$Q = X_q W_Q$$

The flattened support image tokens serve as the key (K) and value (V) pairs:

$$K = \text{ Flatten } \left( \text{ Tokens }_s \right) W_k, V = \text{ Flatten } \left( \text{ Tokens }_s \right) W_V$$

Where $W_k$ and $W_v$ are learned weight matrices for the keys and values, respectively. The crossattention output $X_q^*$ for the query features, influenced by the support image tokens, is computed as:

$$X_q^* = \operatorname{CrossAttn}\left(X_q, \operatorname{Flatten}\left(\text{ Token }_s\right)\right) = \operatorname{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V$$

# References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4432–4441, 2019.

[2] Yuval Alaluf, Daniel Garibi, Or Patashnik, Hadar Averbuch-Elor, and Daniel Cohen-Or. Cross-image attention for zero-shot appearance transfer. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–12, 2024.

[3] Omri Avrahami, Kfir Aberman, Ohad Fried, Daniel Cohen-Or, and Dani Lischinski. Break-a-scene: Extracting multiple concepts from a single image. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–12, 2023.

[4] Omri Avrahami, Thomas Hayes, Oran Gafni, Sonal Gupta, Yaniv Taigman, Devi Parikh, Dani Lischinski, Ohad Fried, and Xi Yin. Spatext: Spatio-textual representation for controllable image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18370–18380, 2023.

[5] Dmitry Baranchuk, Ivan Rubachev, Andrey Voynov, Valentin Khrulkov, and Artem Babenko. Label-efficient semantic segmentation with diffusion models. *arXiv preprint arXiv:2112.03126*, 2021.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[7] Mingdeng Cao, Xintao Wang, Zhongang Qi, Ying Shan, Xiaohu Qie, and Yinqiang Zheng. Masactrl: Tuning-free mutual self-attention control for consistent image synthesis and editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22560–22570, 2023.

[8] Arantxa Casanova, Marlene Careil, Jakob Verbeek, Michal Drozdzal, and Adriana Romero Soriano. Instance-conditioned gan. *Advances in Neural Information Processing Systems*, 34:27517–27529, 2021.

[9] Hila Chefer, Yuval Alaluf, Yael Vinker, Lior Wolf, and Daniel Cohen-Or. Attend-and-excite: Attention-based semantic guidance for text-to-image diffusion models. *ACM Transactions on Graphics (TOG)*, 42(4):1–10, 2023.

[10] Jang Hyun Cho, Utkarsh Mall, Kavita Bala, and Bharath Hariharan. Picie: Unsupervised semantic segmentation using invariance and equivariance in clustering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16794–16804, 2021.

[11] Guillaume Couairon, Jakob Verbeek, Holger Schwenk, and Matthieu Cord. Diffedit: Diffusion-based semantic image editing with mask guidance. *arXiv preprint arXiv:2210.11427*, 2022.

[12] Weixi Feng, Xuehai He, Tsu-Jui Fu, Varun Jampani, Arjun Akula, Pradyumna Narayana, Sugato Basu, Xin Eric Wang, and William Yang Wang. Training-free structured diffusion guidance for compositional text-to-image synthesis. *arXiv preprint arXiv:2212.05032*, 2022.

[13] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022.

[14] Rinon Gal, Moab Arar, Yuval Atzmon, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. Encoder-based domain tuning for fast personalization of text-to-image models. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023.

[15] Sooyeon Go, Kyungmook Choi, Minjung Shin, and Youngjung Uh. Eye-for-an-eye: Appearance transfer with semantic correspondence in diffusion models. *arXiv preprint arXiv:2406.07008*, 2024.

[16] Mark Hamilton, Zhoutong Zhang, Bharath Hariharan, Noah Snavely, and William T Freeman. Unsupervised semantic segmentation by distilling feature correspondences. *arXiv preprint arXiv:2203.08414*, 2022.

[17] Eric Hedlin, Gopal Sharma, Shweta Mahajan, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. Unsupervised semantic correspondence using stable diffusion. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

[19] Susung Hong, Gyuseong Lee, Wooseok Jang, and Seungryong Kim. Improving sample quality of diffusion models using self-attention guidance. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7462–7471, 2023.

[20] Ziqi Huang, Tianxing Wu, Yuming Jiang, Kelvin CK Chan, and Ziwei Liu. Reversion: Diffusion-based relation inversion from images. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024.

[21] Tero Karras. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2019.

[22] Yunji Kim, Jiyoung Lee, Jin-Hwa Kim, Jung-Woo Ha, and Jun-Yan Zhu. Dense text-to-image generation with attention modulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7701–7711, 2023.

[23] Yijun Li, Richard Zhang, Jingwan Lu, and Eli Shechtman. Few-shot image generation with elastic weight consolidation. *arXiv preprint arXiv:2012.02780*, 2020.

[24] Long Lian, Boyi Li, Adam Yala, and Trevor Darrell. Llm-grounded diffusion: Enhancing prompt understanding of text-to-image diffusion models with large language models. *arXiv preprint arXiv:2305.13655*, 2023.

[25] Jishnu Mukhoti, Tsung-Yu Lin, Omid Poursaeed, Rui Wang, Ashish Shah, Philip HS Torr, and Ser-Nam Lim. Open vocabulary semantic segmentation with patch aligned contrastive learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19413–19423, 2023.

[26] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2085–2094, 2021.

[27] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM Transactions on graphics (TOG)*, 42(1):1–13, 2022.

[28] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 22500–22510, 2023.

[29] Junjiao Tian, Lavisha Aggarwal, Andrea Colaco, Zsolt Kira, and Mar Gonzalez-Franco. Diffuse attend and segment: Unsupervised zero-shot segmentation using stable diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3554–3563, 2024.

[30] Omer Tov, Yuval Alaluf, Yotam Nitzan, Or Patashnik, and Daniel Cohen-Or. Designing an encoder for stylegan image manipulation. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021.

[31] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1921–1930, 2023.

[32] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[33] Andrey Voynov, Qinghao Chu, Daniel Cohen-Or, and Kfir Aberman. p+: Extended textual conditioning in text-to-image generation. *arXiv preprint arXiv:2303.09522*, 2023.

[34] Yanchao Yang and Stefano Soatto. Fda: Fourier domain adaptation for semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4085–4095, 2020.

[35] Hu Yu, Hao Luo, Fan Wang, and Feng Zhao. Uncovering the text embedding in text-to-image diffusion models. *arXiv preprint arXiv:2404.01154*, 2024.

[36] Muzhi Zhu, Yang Liu, Zekai Luo, Chenchen Jing, Hao Chen, Guangkai Xu, Xinlong Wang, and Chunhua Shen. Unleashing the potential of the diffusion model in few-shot semantic segmentation. *arXiv preprint arXiv:2410.02369*, 2024.